Visual Component Library Reference Volume 2

Borland®

Delphi™ 3

for Windows 95 & Windows NT

Visual Component Library Reference

VOLUME 2

Borland®
Delphi™
for Windows 95 and Windows NT

Borland International, Inc., 100 Borland Way
P.O. Box 660001, Scotts Valley, CA 95067-0001

Borland may have patents and/or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

COPYRIGHT © 1983, 1997 Borland International. All rights reserved. All Borland product names are trademarks or registered trademarks of Borland International, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

Printed in the U.S.A.

HDA1330WW21774 1E0R0297
9798990001-9 8 7 6 5 4 3 2 1
H1

Contents

TDBMemo 867
TDBNavigator 876
TDBRadioGroup 884
TDBRichEdit 891
TDBText 901
TDdeClientConv 906
TDdeClientItem 917
TDdeServerConv 921
TDdeServerItem 925
TDecisionCube 931
TDecisionGraph 937
TDecisionGrid 944
TDecisionPivot 949
TDecisionQuery 958
TDecisionSource 964
TDirectoryListBox 986
TDrawGrid 996
TDriveComboBox 1007
TDSTableProducer 1013
TEdit 1027
TField 1032
TFileListBox 1065
TFilterComboBox 1074
TFindDialog 1080
TFloatField 1086
TFontDialog 1093
TForm 1099
TGraphicControl 1105
TGraphicField 1110
TGroupBox 1113
THeaderControl 1118
THintWindow 1125
THotKey 1130
TImage 1134
TImageList 1141
TInplaceEdit 1144
TIntegerField 1153
TISAPIApplication 1160
TISAPIApplication 1161
TLabel 1164
TListBox 1167
TListView 1172
TLoginDialog 1177
TMainMenu 1182
TMaskEdit 1187
TMediaPlayer 1191
TMemo 1222
TMemoField 1226
TMenu 1230
TMenuItem 1235
TNumericField 1247
TObject 1255
TOleContainer 1264
TOleControl 1284
TOpenDialog 1291
TOpenPictureDialog 1299
TOpenPictureDialog 1301
TPageControl 1302
TPageProducer 1309
TPaintBox 1312
TPanel 1316
TPasswordDialog 1321
TPersistent 1327
TPopupMenu 1331
TPrintDialog 1336
TPrinterSetupDialog 1341
TProgressBar 1343
TPropertyPage 1349
TProvider 1356
TQuery 1361
TQueryTableProducer 1374
TRadioButton 1378
TRadioGroup 1383
TRemoteServer 1388
TReplaceDialog 1393
TReport 1396
TRichEdit 1408
TRichEdit 1411
TSaveDialog 1413
TSavePictureDialog 1415
TScreen 1418
TScrollBar 1430
TScrollBox 1438
TScrollingWinControl 1443
TServerSocket 1451
TSession 1454
TShape 1476
TSmallIntField 1481
TSpeedButton 1485
TSplitter 1493
TStaticText 1498
TStatusBar 1502
TStoredProc 1508
TStringField 1519

TStringGrid 1526
TStringGrid 1527
TTabControl 1533
TTable 1538
TTabSheet 1558
TTimeField 1563
TTimer 1567
TToolBar 1571
TToolButton 1580
TToolWindow 1586
TTrackBar 1591
TTreeView 1599
TUpdateSQL 1604
TUpDown 1611
TVarBytesField 1615
TWebApplication 1619
TWebDispatcher 1626
TWebModule 1629
TWinControl 1631
TWordField 1675

Index **1681**

TDBMemo

Unit: DBCtrls | Palette: Data Controls

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomEdit • TCustomMemo • TDBMemo

TDBMemo is a multiline edit control that can display and edit a field in a dataset.

Description

Use *TDBMemo* to let users edit a field that may contain lengthy textual data or to simply display the contents of such a field. *TDBMemo* uses the *Text* property to represent the contents of the field.

TDBMemo permits multiple lines of text. Thus, *TDBMemo* is appropriate for long alphanumeric fields or text binary large objects (BLOBs). For short alphanumeric fields, consider using a *TDBEdit* component instead.

If the application doesn't require the data-aware capabilities of *TDBMemo*, use a memo control (*TMemo*) instead, to conserve system resources.

See also

T*DataSet* object, T*DataSource* object, *TDBEdit* object, *TDBLookupCombo* object, *TField* object, *TMemo* object

Properties

The following table lists the properties of *TDBMemo*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
Alignment	published	taLeftJustify	TCustomMemo
AutoDisplay	published	True	TDBMemo
BorderStyle	published	bsSingle	TCustomEdit
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl

Property	Visibility	Default	Documented in
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DataField	published		TDBMemo
DataSource	published		TDBMemo
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Field	public, read-only		TDBMemo
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
ImeMode	published	imDontCare	TWinControl
ImeName	published		TWinControl
Left	published		TControl
Lines	public		TCustomMemo
MaxLength	published	0	TCustomEdit
Modified	public		TCustomEdit
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ReadOnly	published	False	TDBMemo
ScrollBars	published	ssNone	TCustomMemo
SelLength	public		TCustomEdit
SelStart	public		TCustomEdit
SelText	public		TCustomEdit
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent
Text	public		TControl

Property	Visibility	Default	Documented in
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
WantTabs	published	False	TCustomMemo
Width	published		TControl
WindowProc	public		TControl
WordWrap	published	True	TCustomMemo

AutoDisplay

AutoDisplay determines whether to automatically display the contents of a memo or graphic BLOB in a database memo.

property AutoDisplay: Boolean;

Description If *AutoDisplay* is *True*, the control automatically displays new data when the underlying BLOB field changes (such as when moving to a new record).

If *AutoDisplay* is *False*, the control shows only the field name whenever the underlying BLOB field changes. To display the data, the user can double-click on the control or select it and press *Enter*.

The effect of *AutoDisplay* is not purely cosmetic. The data shown by the *TDBMemo* is the value of the *Text* property. When *AutoDisplay* is *False*, if the data changes, the value of the *Text* property changes to the name of the field. Thus, if *AutoDisplay* is *False*, applications should be cautious about using the value of the *Text* property to ascertain the value of the underlying field.

Calling the *LoadMemo* method causes *Text* to update to the current value of the BLOB field. This change will also be reflected in the appearance of the control on screen.

Change the value of *AutoDisplay* to *False* if the automatic loading of BLOB fields takes too long.

Note *AutoDisplay* only applies to BLOB fields. If the control is bound to another type of text field, *AutoDisplay* has no effect.

See also *DataField* property, *Field* property, *KeyPress* method, *LoadMemo* method, *Text* property

DataField

DataField specifies the name of the field for which the database memo displays data.

property DataField: **string**;

Description Use *DataField* to link the memo control to a field in the dataset. Just setting *DataField* by itself will not link the memo control to a database field.

Additionally, the dataset which contains the field must be specified by setting the *DataSource* property.

See also *DataSource* property, *Field* property

DataSource

DataSource links the memo control to the dataset that contains the field for which the memo control displays data.

property DataSource: TDataSource;

Description Link the memo control with a dataset by setting *DataSource* to a data source component that identifies the desired dataset. Specify which field in the dataset contains the memo data using the *DataField* property.

See also *DataField* property

Field

Field specifies the *TField* component the database memo is linked to.

property Field: TField;

Description Read *Field* to get direct access to the contents and properties of the database field without going through the edit window. Use *Field* to change the contents of the database field programmatically.

Use the *Text* property instead to change the contents of the memo control without immediately writing the changes to the record in the dataset. Changing the *Text* property allows the user to cancel the changes.

See also *DataField* property, *Text* property

ReadOnly

ReadOnly determines whether the user can change the contents of the database memo.

property ReadOnly: Boolean;

Description If *ReadOnly* is *True*, the user can't change the contents. If *ReadOnly* is *False*, the user can modify the contents.

The *ReadOnly* property determines whether the user can use the database memo to change the value of the field of the current record, or if the control only displays the value. If *ReadOnly* is *False*, the user can change the field's value as long as the dataset is in edit mode.

See also *Text* property

Events

The following table lists the events of *TDBMemo*:

Event	Visibility	Documented in
OnChange	published	TCustomEdit
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TDBMemo*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
Change	protected	TDBMemo
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TCustomEdit
ClearSelection	public	TCustomEdit
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl

Method	Visibility	Documented in
CopyToClipboard	public	TCustomEdit
Create	public	TDBMemo
CreateParented	public	TWinControl
CutToClipboard	public	TCustomEdit
DefaultHandler	public	TObject
Destroy	public	TDBMemo
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetSelTextBuf	public	TCustomEdit
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
KeyDown	protected	TDBMemo
KeyPress	protected	TDBMemo

Method	Visibility	Documented in
LoadMemo	public	TDBMemo
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Notification	protected	TDBMemo
PaintTo	public	TWinControl
PasteFromClipboard	public	TCustomEdit
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SelectAll	public	TCustomEdit
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetSelTextBuf	public	TCustomEdit
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl
WndProc	protected	TDBMemo

D

Change

Change informs the data source of changes to the value of the text, before invoking the *OnChange* event handler.

procedure Change; **override**;

Description Applications do not call the *Change* method directly. It is triggered automatically by Windows events. Derived classes can override the *Change* method to make additional responses to changes in the *Text* property. Any class that overrides the *Change* method should call the inherited method as well.

See also *OnChange* event

Create

Create creates an instance of *TDBMemo*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to create and initialize an instance of *TDBMemo*. *Create* overrides the inherited constructor to initialize the data handling aspects of a *TDBMemo*.

See also *TCustomMemo.Create* method

Destroy

Destroy destroys an instance of *TDBMemo*.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the *TDBMemo* object is not already freed and only then calls *Destroy*.

Destroy frees up the memory associated with data handling that was allocated in the constructor, before calling the inherited destructor.

See also *Free* method

KeyDown

KeyDown overrides the inherited method to put the dataset in an editing state if the user types *Del* or *Shift+Insert*.

procedure KeyDown(**var** Key: Word; Shift: TShiftState); **override**;

Description KeyDown first calls the inherited method, which calls the *OnKeyDown* event handler. *KeyDown* next attempts to put the dataset into an editing state when the user types *Del* or *Shift+Insert*. If the memo control is not linked to a dataset, *KeyDown* discards the keystroke. Derived classes can override *KeyDown* to handle additional key combinations, but the inherited method should always be called.

See also *DataSource* property, *KeyPress* method, *OnKeyDown* event

KeyPress

KeyPress overrides the inherited method to handle any keystrokes that alter the state of the edit control.

procedure KeyPress(**var** Key: Char); **override**;

Description *KeyPress* first calls the inherited method, which calls the *OnKeyPress* event handler. Next, *KeyPress* checks the value of the key, and throws it away if it is not a valid character for the field type, or if there is no link to a database field. Valid characters cause *KeyPress* to attempt to put the dataset into edit mode.

If the field for the memo control is a text BLOB, pressing the *Pause* key causes *KeyPress* to call the *LoadMemo* method. Pressing the *Escape* key causes *KeyPress* to throw away unposted edits.

See also *AutoDisplay* property, *LoadMemo* method, *OnKeyPress* event

LoadMemo

LoadMemo loads the text BLOB from the field in the database into the memo control.

```
procedure KeyPress(var Key: Char); override;
```

Description *LoadMemo* allows an application to control when the memo control displays the actual contents of a text BLOB field. Use *LoadMemo* to update the text in the memo control to the value of the field in the current record. *LoadMemo* only works with text BLOB fields, when the *AutoDisplay* property is *False*, so that the value of the field is not loaded automatically.

See also *AutoDisplay* property, *Field* property, *OnKeyPress* method, *Text* property

Notification

Notification overrides the inherited method to respond to *opRemove* notifications on the *DataSource*.

```
type TOperation = (opInsert, opRemove);
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

Description After calling the inherited method, *Notification* checks whether the notification indicates the removal of the *DataSource*. If so, *Notification* sets the *DataSource* property to **nil**.

Derived classes can override *Notification* method to handle additional notifications, but the inherited method should be always be called.

See also *DataSource* property

WndProc

WndProc provides specific message responses for the memo control.

```
procedure WndProc(var Message: TMessage); override;
```

Description *WndProc* overrides the *WndProc* for *TWinControl* to handle window messages that are specific to painting a *TDBMemo* control, and sends all others to the inherited *WndProc*.

Derived classes that override *WndProc* to provide specialized responses to messages should call the inherited *WndProc* at the end to dispatch any other messages.

See also *TWinControl.WndProc* property

TDBNavigator

Unit: DBCtrls | Palette: Data Controls

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomControl • TCustomPanel • TDBNavigator

The *TDBNavigator* component (a database navigator) is used to move through the data in a dataset and perform operations on the data, such as inserting a blank record or posting a record.

Description

Use the database navigator on forms that contain data-aware controls, such as *TDBGrid* or *TDBEdit*. *TDBNavigator* provides the user with control over the dataset when editing or viewing the data.

When the user chooses one of the navigator buttons, the appropriate action occurs on the dataset the navigator is linked to. For example, if the user clicks the *Insert* button, a blank record is inserted in the dataset.

TDBNavigator can show any or all of the following buttons:

Button	Purpose
First	Sets the current record to the first record in the dataset, disables the First and Prior buttons, and enables the Next and Last buttons if they are disabled.
Prior	Sets the current record to the previous record and enables the Last and Next buttons if they are disabled.
Next	Sets the current record to the next record and enables the First and Prior buttons if they are disabled.
Last	Sets the current record to the last record in the dataset, disables the Last and Next buttons, and enables the First and Prior buttons if they are disabled.
Insert	Inserts a new record before the current record, and sets the dataset into Insert and Edit states.
Delete	Deletes the current record and makes the next record the current record.
Edit	Puts the dataset into Edit state so that the current record can be modified.
Post	Writes changes in the current record to the database.
Cancel	Cancels edits to the current record, restores the record display to its condition prior to editing, and turns off Insert and Edit states if they are active.
Refresh	Refreshes the buffered data in the associated dataset. For TQuery components, this button is disabled unless the RequestLive property is True.

See also

TDataSource object, *TDataSet* object, *TDBGrid* object, *TDBEdit* object

Properties

The following table lists the properties of *TDBNavigator*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ConfirmDelete	published	True	TDBNavigator
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DataSource	published		TDBNavigator
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Flat	published	False	TDBNavigator
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Hints	published		TDBNavigator
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentCtl3D	published	True	TWinControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl

Property	Visibility	Default	Documented in
TabStop	published	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
VisibleButtons	published	[nbFirst, nbPrior, nbNext, nbLast, nbInsert, nbDelete, nbEdit, nbPost, nbCancel, nbRefresh]	TDBNavigator
Width	published		TControl
WindowProc	public		TControl

ConfirmDelete

ConfirmDelete determines whether a message box appears asking the user to confirm the deletion when the user uses the database navigator to delete the current record in the dataset.

```
property ConfirmDelete: Boolean;
```

Description Use the *ConfirmDelete* property to help prevent the user from accidentally deleting a record from the dataset.

If *ConfirmDelete* is *True*, a message box appears when the user presses the delete button and the record isn't deleted unless the user chooses the OK button. If *ConfirmDelete* is *False*, no message box appears and the record is simply deleted.

See also *DataSource* property, *VisibleButtons* property

DataSource

DataSource links the navigator to the dataset which it controls.

```
property DataSource: TDataSource;
```

Description Use the *DataSource* property to link the database navigator with a dataset. Specify the data source component that identifies the dataset the data is found in.

See also *TDataSet* object, *TDataSource* object, *TNavDataLink* object

Flat

Flat determines whether the buttons of the navigator have a three-dimensional (3-D) look with borders or two-dimensional look without borders.

```
property Flat: Boolean;
```

Description When *Flat* is *True*, the navigator buttons appear flat and do not have borders separating them. When *Flat* is *False*, the buttons are clearly defined.

See also *Ctl3D* property

Hints

Hints provides a way to customize the Help Hints for the buttons on the database navigator.

```
property Hints: TStrings;
```

Description Use the *Hints* property to supply Help Hints of your choosing for the individual navigator buttons. Each button has a default Help Hint. *Hints* allow the values of any or all of these default Help Hints to be replaced by customized hints.

Hints is a string list. Each hint is a string. The first string in the string list becomes the Help Hint for the first button on the navigator (the First button). The seventh hint becomes the Help Hint for the seventh button (the Edit button).

When specifying Hints at runtime, enter an empty string ('') for any Help Hint that should keep the default value. Simply leave the line blank when using the string list property editor of the Object Inspector for the *Hints* property.

Note To have the Help Hints appear at runtime, set the *ShowHint* property to *True*.

See also *ShowHint* property, *TStrings* object

VisibleButtons

The *VisibleButtons* property array determines which buttons appear on the database navigator.

```
type
  TNavigateBtn = (nbFirst, nbPrior, nbNext, nbLast, nbInsert, nbDelete, nbEdit, nbPost,
   nbCancel, nbRefresh);

  TButtonSet = set of TNavigateBtn;

property VisibleButtons: TButtonSet;
```

Description Use *VisibleButtons* to select which buttons appear on the navigator. Leave any of the navigator buttons out of the VisibleButtons set to hide those buttons and thereby prevent the user from performing certain operations. For example, to only allow the user to view the records in the dataset, VisibleButtons should include only the *nbFirst*, *nbPrior*, *nbNext*, and *nbLast*

The following buttons are available for *VisibleButtons*:

Button	Value	Action
First	nbFirst	Go to the first record
Prior	nbPrior	Go to the previous record
Next	nbNext	Go to the next record
Last	nbLast	Go to the last record
Insert	nbInsert	Insert a blank record
Delete	nbDelete	Deletes the current record
Edit	nbEdit	Permits editing of the current record
Post	nbPost	Posts the current record
Cancel	nbCancel	Cancels the current edit
Refresh	nbRefresh	Refreshes the data in the dataset

See also *Hints* property, *Visible* property

Events

The following table lists the events of *TDBNavigator*:

Event	Visibility	Documented in
BeforeAction	published	TDBNavigator
OnClick	published	TDBNavigator
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnResize	published	TCustomPanel
OnStartDrag	published	TControl

BeforeAction

BeforeAction occurs when a button on the database navigator is clicked, before the action is executed.

```
type
  TNavigateBtn = (nbFirst, nbPrior, nbNext, nbLast, nbInsert, nbDelete, nbEdit, nbPost,
  nbCancel, nbRefresh);

ENavClick = procedure (Sender: TObject; Button: TNavigateBtn) of object;

property BeforeAction: ENavClick;
```

Description Write a *BeforeAction* event handler to take specific action before the default response when a user clicks a button in the database navigator.

See also *OnClick* event

OnClick

OnClick occurs when a button on the database navigator is clicked, after the action is executed.

```
type
  TNavigateBtn = (nbFirst, nbPrior, nbNext, nbLast, nbInsert, nbDelete, nbEdit, nbPost,
   nbCancel, nbRefresh);

  ENavClick = procedure (Sender: TObject; Button: TNavigateBtn) of object;

property OnClick: ENavClick;
```

Use the *OnClick* event handler to write code that responds to the click event. Because the buttons on the navigator already have default actions that occur when they are clicked, it isn't necessary to write an *OnClick* event handler.

The *OnClick* event occurs when

- The user clicks one of the buttons on the navigator with the mouse.
- The user presses *Spacebar* while the navigator has focus.
- The *BtnClick* method is called.

Note The *OnClick* event is triggered after the default action for the clicked button occurs.

See also *BeforeAction* event

Methods

The following table lists the methods of *TDBNavigator*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
BtnClick	public	TDBNavigator
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject

Method	Visibility	Documented in
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TDBNavigator
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TDBNavigator
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject

Method	Visibility	Documented in
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TDBNavigator
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

BtnClick

The *BtnClick* method simulates a button click on the database navigator, invoking the action of the button.

```
type
  TNavigateBtn = (nbFirst, nbPrior, nbNext, nbLast, nbInsert, nbDelete, nbEdit, nbPost,
   nbCancel, nbRefresh);

procedure BtnClick(Index: TNavigateBtn);
```

Description Use *BtnClick* when you want the application, not the user, to control the database navigator. Call *BtnClick*, specifying a *TNavigateBtn* type as the value of the Index parameter, to invoke the command associated with the selected button.

Calling *BtnClick* triggers the navigator's *OnClick* event after the default action of the button.

See also *OnClick* event

Create

The *Create* method creates an instance of *TDBNavigator*.

```
constructor Create(AOwner: TComponent); override;
```

Description Call *Create* to instantiate a *TDBNavigator* at runtime. For *TDBNavigator* objects placed on a form or data module at design time, *Create* is called automatically.

After calling the inherited creator, *Create* creates the helper objects used by *TDBNavigator* to manage the buttons and the link to the dataset, and initializes some properties to affect the appearance of the navigator.

See also *Destroy* method, *TCustomPanel.Create* method

Destroy

The *Destroy* method destroys an instance of *TDBNavigator*.

```
destructor Destroy; override;
```

Description Do not call Destroy directly in an application. Instead, call *Free*. *Free* verifies that the *TDBNavigator* object is not already freed and only then calls destroy.

Destroy frees the helper objects that were instantiated in the *Create* method before calling the inherited destructor.

See also *Create* method, *Free* method

SetBounds

The *SetBounds* sets the database navigator's boundary properties all at one time.

```
procedure Setbounds(ALeft, ATop, AWidth, AHeight: Integer);
```

Description Call *SetBounds* to set more than one of the navigator's boundary properties at a time. The *Left*, *Top*, *Width*, and *Height* properties are set to the values passed in the *ALeft*, *ATop*, *AWidth*, and *AHeight* parameters, respectively. *SetBounds* arranges the buttons on the navigator to fit within the boundaries.

Although you can always set the individual boundaries, using *SetBounds* makes several changes at once without rearranging the buttons for each change.

See also *Height* property, *Left* property, *Top* property, *Width* property

TDBRadioGroup

Unit: DBCtrls — Palette: Data Controls

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomControl • TCustomGroupBox • TCustomRadioGroup • TDBRadioGroup

TDBRadioGroup represents a group of radio buttons connected to a database.

Description Use *TDBRadioGroup* to group a set of data-aware radio buttons. Radio buttons that are contained directly in the same control component are said to be grouped; when a radio button is checked, all other radio buttons in its group become unchecked. Database radio groups provide an easy way to ensure that the user enters at most one of the options presented for a field. They are also useful for displaying data from fields that have only a few possible values.

To add radio buttons to a *TDBRadioGroup*, edit the *Items* property in the Object Inspector. Each string in *Items* makes a radio button appear in the group box with the string as its caption.

When the user selects a radio button, the value of the selected button becomes the contents of the linked field for the current record in the dataset. Specify values for the buttons by editing the *Values* property in the Object Inspector; the first string in *Values* corresponds to the first string in *Items*, and so forth. If no *Values* strings are set, each button's caption serves as its value.

See also *TDataSource* object, *TRadioButton* object

Properties

The following table lists the properties of *TDBRadioGroup*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Caption	published		TControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
Columns	published	1	TCustomRadioGroup
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl

Property	Visibility	Default	Documented in
Cursor	published	crDefault	TControl
DataField	published		TDBRadioGroup
DataSource	published		TDBRadioGroup
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Field	public, read-only		TDBRadioGroup
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
ItemIndex	public	–1	TCustomRadioGroup
Items	published		TDBRadioGroup
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	True	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ReadOnly	published	False	TDBRadioGroup
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
Value	public		TDBRadioGroup
Values	published		TDBRadioGroup
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

DataField

The field from which the radio group displays data.

property DataField: **string**;

Description Use *DataField* to specify the name of the database field that the radio group will represent.

See also *DataSource* property

DataSource

Determines where the database radio group obtains the data to display.

property DataSource: TDataSource;

Description *DataSource* holds a *TDataSource* object that points to a dataset component, such as a table. The dataset component should be linked to the database you want to access.

See also *DataField* property, *TDataSource* object, *TDataSet* object, *TDBDataSet* object, *TTable* object

Field

The *TField* component that the radio group is linked to.

property Field: TField;

Description *Field* holds the *TField* object that the linked dataset component creates to access the database.

See also *TField* object, *TDataSet* object, *TDBDataSet* object

Items

A list of the radio buttons in the radio group.

property Items: TStrings;

Description *Items* holds a *TStrings* object that lists the captions of the radio buttons in the group. (To add or remove radio buttons, edit the *Items* list from the Object Inspector.) These captions become the values of the radio buttons, unless you override them in the *Values* property; when a button is selected, its value becomes the content of the linked field for the current record in the database.

See also *TCustomRadioGroup.Items* property, *ItemIndex* property, *Values* property

ReadOnly

Determines whether the user can select a different radio button in the group, and hence modify the data in the field.

property ReadOnly: Boolean;

Description If *ReadOnly* is *True*, the radio group displays data from the current record, but cannot be used to change the contents of the field. The default is *False*.

See also *CanModify* method

Value

The content of the linked field for the current record in the database.

property Value: string;

Description *Value* holds the content of the linked field for the current record. If the user selects a different radio button, the *Value* property changes.

Values

Overrides the buttons' captions to determine the values of the radio buttons.

property Values: TStrings;

Description When the user selects a radio button, the value of that button is written to the linked field in the database. By default, the value of a button is simply the caption that appears next to it on the screen, as determined by the *Items* property.

In some cases, you may want the values of the radio buttons to differ from their captions. For example, if you use radio buttons to represent a database field whose content can be "Y" or "N", you may want the buttons' captions to be "Yes" and "No." In this case, you would enter "Yes" and "No" in the *Items* list, and enter "Y" and "N" in the *Values* list.

You can edit the *Values* list in the Object Inspector, just as you would the *Items* list. To keep the buttons' captions as their values, leave the *Values* property empty.

See also *Items* property

Events

The following table lists the events of *TDBRadioGroup*:

Event	Visibility	Documented in
OnChange	published	TDBRadioGroup
OnClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnStartDrag	published	TControl

OnChange

OnChange occurs when the user changes the data in the linked field.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnChange: TNotifyEvent
```

Description The *OnChange* event occurs when a change is made to the contents of the database field accessed by the radio group. Use an *OnChange* event handler to implement any special processing that should be executed when the radio group is updated.

See also *TNotifyEvent* type

Methods

The following table lists the methods of *TDBRadioGroup*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
CanModify	protected	TDBRadioGroup
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TDBRadioGroup
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TDBRadioGroup
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl

Method	Visibility	Documented in
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl

Method	Visibility	Documented in
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

CanModify

Determines whether the user can select a different radio button in the group.

function CanModify: Boolean; **override**;

Description *CanModify* returns *True* if the radio group is linked to an editable data source and *ReadOnly* is set to *False*.

See also *DataSource* property, *TCustomRadioGroup.CanModify* method, *ReadOnly* property

Create

Creates and initializes a *TDBRadioGroup* instance.

constructor Create(AOwner: TComponent); **override**;

Description *Create* calls the inherited constructor method, then initializes internal objects and variables used to control data access.

Destroy

Destroys the *TDBRadioGroup* instance and the radio buttons it contains.

destructor Destroy; **override**;

Description *Destroy* frees internally created objects that control data access, then calls the inherited destructor method.

TDBRichEdit

Unit: DBCtrls Palette: Data Controls

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomEdit • TCustomMemo • TCustomRichEdit • TDBRichEdit

A *TDBRichEdit* object is a multiline edit control that can display and edit a rich text memo field in a dataset.

Description Use a *TDBRichEdit* object to enable users to edit a database field in a rich text edit control. *TDBRichEdit* provides the properties and methods to enter and

work with rich text. *TRichEdit* does not provide any user interface components to make these formatting options available to the user. Applications must implement the user interface components to surface the rich text capabilities of a *TDBRichEdit* object.

TDBRichEdit uses the *Text* property to represent the contents of the field.

If the database field does not include text formatting information, consider using a *TDBMemo* object or a *TDBEdit* object instead.

If the application doesn't require the data-aware capabilities of *TDBRichEdit*, use a rich text edit control (*TRichEdit*) instead, to conserve system resources.

See also *TDataSet* object, *TDataSource* object, *TDBEdit* object, *TDBMemo* object, *TField* object, *TRichEdit* object

Properties

The following table lists the properties of *TDBRichEdit*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
Alignment	published	taLeftJustify	TCustomMemo
AutoDisplay	published	True	TDBRichEdit
BorderStyle	published	bsSingle	TCustomEdit
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DataField	published		TDBRichEdit
DataSource	published		TDBRichEdit

Property	Visibility	Default	Documented in
DefAttributes	public		TCustomRichEdit
DefaultConverter	public		TCustomRichEdit
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Field	public, read-only		TDBRichEdit
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
HideScrollBars	published	True	TCustomRichEdit
HideSelection	published	True	TCustomRichEdit
Hint	published		TControl
ImeMode	published	imDontCare	TWinControl
ImeName	published		TWinControl
Left	published		TControl
Lines	public		TCustomMemo
MaxLength	published	0	TCustomEdit
Modified	public		TCustomEdit
Name	published		TControl
Owner	public, read-only		TComponent
PageRect	public		TCustomRichEdit
Paragraph	public, read-only		TCustomRichEdit
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PlainText	published	False	TCustomRichEdit
PopupMenu	published		TControl
ReadOnly	published	False	TDBRichEdit
ScrollBars	published	ssNone	TCustomMemo
SelAttributes	public		TCustomRichEdit
SelLength	public		TCustomRichEdit
SelStart	public		TCustomRichEdit
SelText	public		TCustomRichEdit
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl

Property	Visibility	Default	Documented in
Tag	published	0	TComponent
Text	public		TControl
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
WantReturns	published	True	TCustomMemo
WantTabs	published	False	TCustomMemo
Width	published		TControl
WindowProc	public		TControl
WordWrap	published	True	TCustomMemo

AutoDisplay

AutoDisplay determines whether to automatically display the contents of the memo field in the rich edit control.

property AutoDisplay: Boolean;

Description If *AutoDisplay* is *True*, the rich edit control automatically displays new data when the underlying BLOB field changes (such as when moving to a new record).

If *AutoDisplay* is *False*, the control shows only the field name whenever the underlying BLOB field changes. To display the data, the user can double-click on the control or select it and press *Enter*.

The effect of *AutoDisplay* is not purely cosmetic. The data shown by the *TDBRichEdit* object is the value of the *Text* property. When *AutoDisplay* is *False*, if the data changes, the value of the *Text* property changes to the name of the field. Thus, if *AutoDisplay* is *False*, applications should be cautious about using the value of the *Text* property to ascertain the value of the underlying field.

Calling the *LoadMemo* method causes *Text* to update to the current value of the BLOB field. This change will also be reflected in the appearance of the control on screen.

Change the value of *AutoDisplay* to *False* if the automatic loading of BLOB fields seems to take too long.

Note *AutoDisplay* only applies to BLOB fields. If the control is bound to another type of text field, *AutoDisplay* has no effect.

See also *DataField* property, *Field* property, *KeyPress* method, *LoadMemo* method, *Text* property

DataField

DataField specifies the field from which the rich edit control displays data.

property DataField: string;

Description Use *DataField* to bind the rich edit control to a field in the dataset. To fully specify a database field, both the dataset and the field within that dataset must be defined. The *DataSource* property of the rich edit control specifies the dataset which contains the *DataField*.

See also *DataSource* property, *Field* property

DataSource

DataSource links the rich edit control to the dataset that contains the field it represents.

property DataSource: TDataSource;

Description Use *DataSource* to link the rich edit control to a dataset in which the data can be found. To fully specify a database field for the edit control, both the dataset and a field within that dataset must be defined. Use the *DataField* property to specify the particular field within the dataset.

See also *DataField* property, *Field* property

Field

Field specifies the *TField* object for the database field the rich edit control represents.

property Field: TField;

Description Read *Field* to get direct access to the contents and properties of the database field without going through the edit window. Use *Field* to change the contents of the database field programmatically. Use the *Text* property instead if you want to change the contents of the edit window without immediately posting the changes to the dataset. Changing the *Text* property allows the user to cancel the changes.

See also *DataField* property, *DataSource* property, *Reset* method, *Text* property, *TField* object

ReadOnly

ReadOnly determines whether the user can use the rich database edit control to change the value of the field in the current record.

property ReadOnly: Boolean;

Description Set *ReadOnly* to *True* to prevent the user from changing the contents of the database field. When *ReadOnly* is *True*, the edit control is used only to display the contents of the field. If *ReadOnly* is *False*, the user can change the field's value as long as the dataset is in edit mode.

See also *Change* method, *EditCanModify* method, *Field* property

Events

The following table lists the events of *TDBRichEdit*:

Event	Visibility	Documented in
OnChange	published	TCustomEdit
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnProtectChange	published	TCustomRichEdit
OnResizeRequest	published	TCustomRichEdit
OnSaveClipboard	published	TCustomRichEdit
OnSelectionChange	published	TCustomRichEdit
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TDBRichEdit*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
Change	protected	TDBRichEdit
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TCustomRichEdit

Method	Visibility	Documented in
ClearSelection	public	TCustomEdit
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
CopyToClipboard	public	TCustomEdit
Create	public	TDBRichEdit
CreateParented	public	TWinControl
CutToClipboard	public	TCustomEdit
DefaultHandler	public	TObject
Destroy	public	TDBRichEdit
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
FindText	public	TCustomRichEdit
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetSelTextBuf	public	TCustomRichEdit
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent

Method	Visibility	Documented in
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
KeyDown	protected	TDBRichEdit
KeyPress	protected	TDBRichEdit
LoadMemo	public	TDBRichEdit
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Notification	protected	TDBRichEdit
PaintTo	public	TWinControl
PasteFromClipboard	public	TCustomEdit
Perform	public	TControl
Print	public	TCustomRichEdit
Realign	public	TWinControl
Refresh	public	TControl
RegisterConversionFormat	public	TCustomRichEdit
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SelectAll	public	TCustomEdit
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetSelTextBuf	public	TCustomEdit
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

Change

Change keeps track of the fact that the text has been modified, before letting the inherited method call the *OnChange* event handler if it is assigned.

procedure Change; **override**;

Description *Change* is a protected method. Applications cannot call it directly. It is triggered automatically by Windows' events. *Change* responds to these

events by internally keeping track of the fact that the text has been modified. This information can then be used to know whether there are any changes to post to the dataset or to remove the changes if the user requests a reset.

See also *OnChange* event, *Reset* method, *Text* property

Create

Create creates an instance of *TDBRichEdit*.

```
constructor Create(AOwner: TComponent); override;
```

Description Call *Create* to instantiate a *TDBRichEdit* object at runtime. For *TDBRichEdit* objects placed on forms or data modules at design time, *Create* is called automatically.

After calling the inherited constructor to allocate and initialize a generic rich edit object, *Create*

- Sets *ReadOnly* to *True*.
- Sets *AutoDisplay* to *True*.
- Creates and initializes the helper objects needed to link to a dataset.

See also *AutoDisplay* property, *ReadOnly* property, *TCustomRichEdit.Create* method, *TFieldDataLink* object

Destroy

Destroy destroys an instance of *TDBRichEdit*.

```
destructor Destroy; override;
```

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the *TDBRichEdit* object is not already freed and only then calls *Destroy*.

Destroy frees the helper objects of the *TDBRichEdit* before calling the inherited destructor.

See also *Create* method, *Free* method

KeyDown

KeyDown supplements the inherited method to drive the dataset into edit mode when the user presses *Del* or *Shift+Insert*.

```
type TShiftState = set of (ssShift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle, ssDouble);

procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

Description *KeyDown* first calls the inherited method, which calls the *OnKeyDown* event handler. *KeyDown* next attempts to put the dataset into an editing state when the user types *Del* or *Shift+Insert*. Derived classes can override *KeyDown* to

handle additional key combinations, but the inherited method should always be called.

Note *KeyDown* is a protected method. Applications cannot call it directly. It is called automatically in response to Windows events.

See also *DataSource* property, *KeyPress* method, *OnKeyDown* event, *TFieldDataLink.Edit* method

KeyPress

KeyPress supplements the inherited method to handle any keystrokes that alter the state of the edit control.

```
procedure KeyPress(var Key: Char); override;
```

Description *KeyPress* first calls the inherited method, which calls the *OnKeyPress* event handler. Next, *KeyPress* checks the value of the key, and throws it away if it is not a valid character for the field type, or if there is no link to a database field. Valid characters cause *KeyPress* to attempt to put the dataset into edit mode.

Pressing the *Escape* key causes *KeyPress* to throw away all unposted edits. *Pressing* the *Pause* key causes *KeyPress* to call *LoadMemo* to load the value of the memo field associated with the rich edit control.

Note *KeyPress* is a protected method. Applications cannot call it directly. It is called automatically in response to Windows events.

See also *DataSource* property, *KeyDown* method, *LoadMemo* method, *OnKeyPress* event, *TFieldDataLink.Reset* method

LoadMemo

LoadMemo loads the memo BLOB from the field in the database into the rich edit control.

```
procedure LoadMemo;
```

Description *LoadMemo* allows an application to control when the rich edit control displays the actual contents of the memo field. Use *LoadMemo* to update the text in the rich edit control to the value of the field in the current record. *LoadMemo* only works with BLOB fields when the *AutoDisplay* property is *False*, so that the value of the field is not loaded automatically.

See also *AutoDisplay* property, *Field* property, *OnKeyPress* method, *Text* property

Notification

Notification overrides the inherited method to respond to *opRemove* notifications on the *DataSource*.

```
type TOperation = (opInsert, opRemove);

procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

Description After calling the inherited method, *Notification* checks whether the notification indicates the removal of the *DataSource*. If so, *Notification* sets the *DataSource* property to **nil**.

Derived classes can override *Notification* method to handle additional notifications, but the inherited method should be always be called.

Note Notification is a protected method. Applications cannot call it directly.

See also *DataSource* property

TDBText

Unit: DBCtrls | Palette: Data Controls

TObject • TPersistent • TComponent • TControl • TGraphicControl • TCustomLabel • TDBText

TDBText is a data-aware control that displays the value of a field on a form.

Description Use *TDBText* to display the contents of a field in the current record of a dataset on a form. Field values displayed by database text controls cannot be modified by the user using the text control. To allow the user to edit the field value, use *TDBEdit* or *TDBMemo* instead.

If the application doesn't require the data-aware capabilities of *TDBText*, use the label component (*TLabel*) or the static text component (*TStaticText*) instead to conserve system resources.

See also *TDBEdit* object, *TLabel* object, *TDBMemo* object, *TStaticText* object

Properties

The following table lists the properties of *TDBText*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
Alignment	published	taLeftJustify	TCustomLabel
AutoSize	published	False	TDBText
BoundsRect	public		TControl
Canvas	public, read-only		TGraphicControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TControl
ClientRect	public, read-only		TControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent

Property	Visibility	Default	Documented in
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DataField	published		TDBText
DataSource	published		TDBText
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Field	public, read-only		TDBText
Font	published		TControl
Height	published		TControl
Hint	published		TControl
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	True	TControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
PopupMenu	published		TControl
ShowHint	published		TControl
Tag	published	0	TComponent
Top	published		TControl
Transparent	published	False	TCustomLabel
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl
WordWrap	published	False	TCustomLabel

AutoSize

AutoSize determines whether the database text control automatically resizes to the width and length of the current string.

property AutoSize: Boolean;

Description Use *AutoSize* to make the database text control adjust its size automatically so the client area accommodates the height and width of the field value. When

AutoSize is *False*, the control is fixed in size. When *AutoSize* is *True*, the size of the control is readjusted whenever its caption changes. The size of the control is also readjusted when the *Font* property changes.

When *WordWrap* is *True*, the width of the database text control is fixed. If *AutoSize* is also *True*, changes to the field value cause the control to change in height. When *AutoSize* is *True* and *WordWrap* is *False*, the height of the control is determined by the font, and changes to the field value cause the control to change in width.

See also *Font* property, *WordWrap* property

DataField

DataField identifes the field whose value is displayed by the database text control.

```
property DataField: string;
```

Description Use *DataField* to bind the database text control to a field in the dataset. To fully specify a database field, both the dataset and the field within that dataset must be defined. The *DataSource* property of the control specifies the dataset that contains the *DataField*.

See also *DataSource* property, *Field* property

DataSource

DataSource links the text control to the dataset that contains the field it represents.

```
property DataSource: TDataSource;
```

Description Use *DataSource* to link the database text control to a dataset in which the data can be found. To fully specify a database field for the control, both the dataset and a field within that dataset must be defined. Use the *DataField* property to specify the particular field within the dataset.

See also *DataField* property, *TDataSource* object

Field

Field specifies the *TField* object for the database field the database text control represents.

```
property Field: TField;
```

Description Use the *Field* object to change the value of the data in the field programmatically.

See also *TField* object, *DataField* property

Events

The following table lists the events of *TDBText*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TDBText*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
Create	public	TDBText
DefaultHandler	public	TObject
Destroy	public	TDBText
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject

Method	Visibility	Documented in
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTextBuf	public	TControl
GetTextLen	public	TControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
Invalidate	public	TControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Perform	public	TControl
Refresh	public	TControl
RemoveComponent	public	TComponent
Repaint	public	TControl
SafeCallException	public	TComponent
ScreenToClient	public	TControl
SendToBack	public	TControl
SetBounds	public	TControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TControl

Create

Create creates an instance of *TDBText*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to instantiate a *TDBText* object at runtime. For *TDBText* objects placed on forms or data modules at design time, *Create* is called automatically.

After calling the inherited constructor to allocate and initialize a generic label object, *Create*

- Adds *csReplicatable* to *ControlStyle*.

- Sets *AutoSize* to *False*.
- Creates and initializes the helper objects needed to link to a dataset.

See also *AutoSize* property, *ControlStyle* property, *TCustomLabel.Create* method, *TFieldDataLink* object

Destroy

Destroy destroys an instance of *TDBText*.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the *TDBText* object is not already freed and only then calls *Destroy*.

Destroy frees the helper objects of the *TDBText* before calling the inherited destructor.

See also *Create* method, *Free* method

TDdeClientConv

Unit: DDEMan | Palette: System

TObject • TPersistent • TComponent • TDdeClientConv

A *TDdeClientConv* component establishes a DDE conversation with a DDE server application.

Description Use *TDdeClientConv* in conjunction with a *TDdeClientItem* object to enable an application to act as a DDE client. *TDdeClientConv* provides many properties, events, and methods that provide services to

- Link to a specified DDE server with a specified DDE topic.
- Send data or macro commands to the DDE server after a link has been established.
- Specify a format for the data that is sent to the DDE server.
- Allow event handlers to respond when the DDE link is opened or closed.

TDdeClientItem objects are associated with a *TDdeClientConv* by setting their *DdeConv* property.

See also *TDdeClientItem* object, *TDdeServerConv* object, *TDdeServerItem* object

Properties

The following table lists the properties of *TDdeClientConv*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ConnectMode	published	ddeAutomatic	TDdeClientConv
Conv	public, read-only		TDdeClientConv
DataMode	public		TDdeClientConv
DdeFmt	public, read-only		TDdeClientConv
DdeService	published		TDdeClientConv
DdeTopic	published		TDdeClientConv
DesignInfo	public		TComponent
FormatChars	published	False	TDdeClientConv
Name	published		TComponent
Owner	public, read-only		TComponent
ServiceApplication	published		TDdeClientConv
Tag	published	0	TComponent
VCLComObject	public		TComponent
WaitStat	public, read-only		TDdeClientConv

ConnectMode

ConnectMode determines whether the link to the DDE server is established automatically when the form is opened.

```
type TDataMode = (ddeAutomatic, ddeManual);

property ConnectMode: TDataMode;
```

Description Use *ConnectMode* to change the way a DDE conversation is established. If *ConnectMode* is *ddeAutomatic*, the link is established when it is defined by *SetLink*. If *ConnectMode* is *ddeManual*, the application must call the *OpenLink* method or the *PasteLink* method to establish the DDE conversation.

See also *OpenLink* method, *PasteLink* method, *SetLink* method

Conv

Conv is the handle for the current DDE conversation.

```
property Conv: HConv;
```

Description Read *Conv* to get a handle to the DDE conversation when it is required for a Windows API function call. *Conv* is available once a DDE conversation has been established by *OpenLink* or *PasteLink*.

See also *OpenLink* method, *PasteLink* method

DataMode

DataMode provides an interface for a property that governs whether portions of the DDE conversation execute automatically.

```
type TDataMode = (ddeAutomatic, ddeManual);
property DataMode: TDataMode;
```

Description *DataMode* is not implemented for *TDdeClientConv*. Derived objects can use this property to automatically execute portions of a DDE conversation, such as a *RequestData* or *PokeData* call.

Note To determine the way the DDE conversation is established, use *ConnectMode* instead.

See also *ConnectMode* property, *PokeData* method, *RequestData* method

DdeFmt

DdeFmt is an integer value representing the standard clipboard format used for submitting or requesting a data item.

```
property DdeFmt: Integer;
```

Description Use *DdeFmt* to specify the format of DDE data items when it has not been specified by the server.

See also *ServiceApplication* property, *TDdeClientItem* object

DdeService

DdeService specifies the DDE server application to be linked to a DDE client.

```
property DDEService: string;
```

Description Use *DdeService* to specify the server application to link to. Typically, *DDEService* is the file name (with path, if necessary) of the DDE server application's main executable file without the.EXE extension. If the DDE server is an Delphi application, *DDEService* is the project name without the .DPR or .EXE extension. For example, to link to a *TDdeServerConv* component in PROJ1.DPR, set *DDEService* to PROJ1.

At design time, specify *DDEService* by typing the DDE server application name in the Object Inspector or by choosing Paste Link in the DDE Info dialog box. To establish a link at runtime, specify the service and topic with the *SetLink* method.

Note The necessary value for *DdeService* depends on the individual server application. See the documentation of the DDE server application for specific information about how to specify the server application.

See also *DdeTopic* property, *ServiceApplication* property, *SetLink* method, *TDdeServerConv* object

DdeTopic

DdeTopic specifies the topic of a DDE conversation

```
property DDETopic: string;
```

Description Use *DdeTopic* to indicate the server-defined topic name of the DDE conversation. Typically, *DdeTopic* is a file name (with path, if necessary) used by the server application. If the DDE server is a Delphi application, by default *DdeTopic* is the caption of the form containing the linked component. If the DDE client is linked to a *TDdeServerConv* component, *DdeTopic* is the name of the server conversation component rather than the form caption.

At design time, specify *DdeTopic* either by typing the *DdeTopic* name in the Object Inspector or by choosing Paste Link in the DDE Info dialog box. To establish a link at runtime, specify the service and topic with the *SetLink* method.

Note The server application defines the topic names available for a DDE conversation. See the documentation for the DDE server application for specific information about the names of available topics.

See also *DdeService* property, *SetLink* method, *TDdeServerConv* object

FormatChars

FormatChars determines if certain characters are filtered out of text data transferred from a DDE server application.

```
property FormatChars: Boolean;
```

Description Use *FormatChars* with DDE server applications that transfer backspaces, linefeeds, carriage returns, and tabs with the text data. These formatting characters can cause incorrect spacing, line breaks, or characters in the DDE client data. To prevent the formatting characters from interfering with the formatting of the client data, set *FormatChars* to *False*. To include the formatting characters sent by the DDE server, set *FormatChars* to *True*.

See also *RequestData* method

ServiceApplication

ServiceApplication specifies the main executable file name (with path, if necessary) of a DDE server application, without the .EXE extension.

```
property ServiceApplication: string;
```

Description Set *ServiceApplication* to the name of the server application. Typically, this is the same value as the *DdeService* property. Sometimes, however, *DDEService* is a value other than the DDE server application's executable file name.

If the server application is not running when the *TDdeClientConv* tries to establish a DDE conversation, *OpenLink* uses the *ServiceApplication* property to launch the server.

See also *DdeService* property, *OpenLink* method

WaitStat

WaitStat indicates whether the server is in the process of servicing a DDE macro that prohibits asynchronous DDE transactions.

```
property WaitStat: Boolean;
```

Description Read *WaitStat* to learn whether a DDE transaction can be started. When *WaitStat* is *True*, the server is processing the last macro, and attempts to execute a macro, request data, or poke data will fail. When *WaitStat* is *False*, the server is free to execute the next transaction in the DDE conversation.

See also *ExecuteMacro* method, *ExecuteMacroLines* method, *PokeData* method, *PokeDataLines* method, *RequestData* method

Events

The following table lists the events of *TDdeClientConv*:

Event	Visibility	Documented in
OnClose	published	TDdeClientConv
OnOpen	published	TDdeClientConv

OnClose

OnClose occurs when the DDE conversation terminates for any reason.

```
type TNotifyEvent = procedure(Sender: TObject) of object;
```

```
property OnClose: TNotifyEvent;
```

Description Write an *OnClose* event handler to take specific action when the DDE conversation ends. The DDE conversation will end if the server terminates the conversation, if the link is closed or changed, or if the *TDdeClientConv* object is destroyed.

See also *CloseLink* method, *Destroy* method, *SetLink* method

OnOpen

OnOpen occurs immediately after the DDE link to the server has been established.

```
type TNotifyEvent = procedure(Sender: TObject) of object;
```

property OnOpen: TNotifyEvent;

Description Write an *OnOpen* event handler to take specific action whenever a link is established to the DDE server. The DDE link is established by the *OnOpen* method or the *PasteLink* method. *SetLink* will also establish a DDE link if the *ConnectMode* is *ddeAutomatic*. *OnOpen* will not be called if the link fails, such as when the server application can't be launched.

See also *ConnectMode* property, *OpenLink* method, *PasteLink* method, *SetLink* method

Methods

The following table lists the methods of *TDdeClientConv*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
CloseLink	public	TDdeClientConv
Create	public	TDdeClientConv
DefaultHandler	public	TObject
Destroy	public	TDdeClientConv
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
ExecuteMacro	public	TDdeClientConv
ExecuteMacroLines	public	TDdeClientConv
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject

Method	Visibility	Documented in
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
OpenLink	public	TDdeClientConv
PasteLink	public	TDdeClientConv
PokeData	public	TDdeClientConv
PokeDataLines	public	TDdeClientConv
RemoveComponent	public	TComponent
RequestData	public	TDdeClientConv
SafeCallException	public	TComponent
SetLink	public	TDdeClientConv
StartAdvise	public	TDdeClientConv

CloseLink

CloseLink terminates an ongoing DDE conversation.

```
function CloseLink;
```

Description Use *CloseLink* when all communication with the DDE server is finished. After a link is closed, no DDE communication can take place between the DDE client and server until another link is opened.

See also *OnClose* event, *OpenLink* method

Create

Create creates an instance of *TDdeClientConv*.

```
constructor Create(AOwner: TComponent); override;
```

Description Call *Create* to create an instance of *TDdeClientConv*. After calling the inherited constructor, Create creates the helper objects used to manage individual DDE client items.

See also *TComponent.Create* method, *TDdeClientItem* object

Destroy

Destroy destroys an instance of *TDdeClientConv*.

```
destructor Destroy; override;
```

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the *TDdeClientConv* object is not already freed, and only then calls *Destroy*.

Destroy closes the link to the server and frees up all helper objects before calling the inherited destructor.

See also *CloseLink* method

ExecuteMacro

ExecuteMacro sends a macro command string to the DDE server application.

```
function ExecuteMacro(Cmd: PChar; WaitFlg: Boolean): Boolean;
```

Description Call *ExecuteMacro* to send a single macro command to the server application. To send multiple macro commands, use *ExecuteMacroLines* instead. *ExecuteMacro* returns *True* if the macro was successfully passed to the DDE server application. If *ExecuteMacro* was unable to send a command string, *ExecuteMacro* returns *False*.

Cmd is a null-terminated string that contains the macro to be executed by the DDE server application. The actual value of *Cmd* depends on the DDE server application. See the documentation of the DDE server application for the command strings it will accept.

WaitFlg determines if this DDE client should wait until the DDE server application finishes executing the macro before allowing another DDE transaction to succeed. If *WaitFlg* is set to *True*, subsequent calls to *ExecuteMacro*, *ExecuteMacroLines*, *PokeData*, *PokeDataLines*, and *RequestData* will fail until the DDE server application completes the macro.

Attempting to execute a macro or poke data before a DDE server application completes a currently executing macro may cause the executing macro to fail or to produce unpredictable results. See the documentation of the DDE server application for the results of sending command strings or poking data before macro execution has completed.

Note *ExecuteMacro* returns *True* if the macro command was successfully passed to the DDE server. A *True* value does not ensure that the macro command will execute successfully once it has been accepted by the server.

See also *ExecuteMacroLines* method, *PokeData* method, *PokeDataLines* method, *RequestData* method, *WaitStat* property

ExecuteMacroLines

ExecuteMacroLines sends a list of several macro commands to a DDE server application.

```
function ExecuteMacroLines(Cmd: TStrings; WaitFlg: Boolean): Boolean;
```

Description Call *ExecuteMacroLines* to send several macro commands to the server application. To send a single macro command, use *ExecuteMacro* instead. *ExecuteMacroLines* returns *True* if the macro list was successfully passed to the DDE server application. If *ExecuteMacroLines* was unable to send a list of command strings, it returns *False*.

Cmd contains the list of macros to be executed by the DDE server application. *WaitFlg* determines if this DDE client should wait until the DDE server application finishes executing all the macros before allowing another DDE transaction to be sent. If *WaitFlg* is set to *True*, subsequent calls to *ExecuteMacro*, *ExecuteMacroLines*, *PokeData*, *PokeDataLines*, and *RequestData* will fail until the DDE server application completes all of the macros.

See also *ExecuteMacro* method, *PokeData* method, *PokeDataLines* method, *RequestData* method, *TStrings* object, *WaitStat* property

OpenLink

OpenLink initiates a new DDE conversation.

```
function OpenLink: Boolean;
```

Description Call *OpenLink* to establish a new DDE conversation with the *DdeService*. If the server application is not running, *OpenLink* will launch the *ServiceApplication*. If the conversation is successfully opened, an *OnOpen* event occurs and *OpenLink* returns *True*. If the conversation isn't successfully opened, *OpenLink* returns *False*.

Note If *ConnectMode* is *ddeAutomatic*, *OpenLink* is called automatically when the *DdeTopic* and *DdeService* are specified by *SetLink*.

See also *ConnectMode* property, *DdeTopic* property, *DdeService* property, *OnOpen* event, *PasteLink* method, *ServiceApplication* property, *SetLink* method

PasteLink

PasteLink opens a DDE conversation with the current Clipboard object.

```
function PasteLink: Boolean;
```

Description Call *PasteLink* to establish a DDE link to an object on the Clipboard. *PasteLink* sets the *DdeTopic* or *DdeService* properties from the Clipboard. *PasteLink* returns *True* if the link is successfully established.

See also *DdeTopic* property, *DdeService* property, *OpenLink* method

PokeData

PokeData sends data to a DDE server application.

```
function PokeData(Item: string; Data: PChar): Boolean;
```

Description Use *PokeData* to transfer text data to a DDE server that supports poked data. *Item* specifies the linked item in the DDE server. *Data* is a null-terminated string that specifies the text data to transfer.

The value of the DDE item depends on the linked DDE server application. *Item* is typically a selectable portion of text, such as a spreadsheet cell or a

database field in an edit box. If the DDE server is an Delphi application, *Item* is the name of the linked DDE server component.

The usual direction of data flow is from the DDE server to the DDE client application. Some DDE server applications won't accept poked data. *PokeData* returns *True* if the data was successfully transferred, *False* if the data was not successfully transferred.

To poke a string list rather than a single string, use the *PokeDataLines* method.

Note If either the *ExecuteMacro* or *ExecuteMacroLines* method was called with its *WaitFlg* parameter set to *True*, *PokeData* will automatically fail until the server application has completed executing the macro. The *WaitStat* property indicates when *PokeData* will attempt to poke data to the server.

See also *ExecuteMacro* method, *ExecuteMacroLines* method, *PokeDataLines* method, *WaitStat* property

PokeDataLines

PokeDataLines sends multiple lines of data to a DDE server application.

```
function PokeDataLines(Item: string; Data: TStrings): Boolean;
```

Description Use *PokeDataLines* to transfer multiple lines of text data to a DDE server that supports poked data. *Item* specifies the linked item in the DDE server. *Data* is a set of strings that specifies the text data to transfer.

The value of the DDE item depends on the linked DDE server application. *Item* is typically a selectable portion of text, such as a spreadsheet cell or a database field in an edit box. If the DDE server is an Delphi application, *Item* is the name of the linked DDE server component.

The usual direction of data flow is from the DDE server to the DDE client application. Some DDE server applications won't accept poked data. *PokeData* returns *True* if the data was successfully transferred, *False* if the data was not successfully transferred.

To poke a single string rather than a string list, use the *PokeData* method.

Note If either the *ExecuteMacro* or *ExecuteMacroLines* method was called with its *WaitFlg* parameter set to *True*, *PokeData* will automatically fail until the server application has completed executing the macro. The *WaitStat* property indicates when *PokeData* will attempt to poke data to the server.

See also *ExecuteMacro* method, *ExecuteMacroLines* method, *PokeData* method, *TStrings* object, *WaitStat* property

RequestData

RequestData requests data from a DDE server.

```
function RequestData(const Item: string): PChar;
```

Description Call *RequestData* to send a request to the DDE server for the data named by *Item*. *RequestData* returns the value of the named DDE server item. *RequestData* automatically allocates memory to store the returned data, but applications must dispose of this PChar string after processing it. This is done with the *StrDispose* function.

If the DDE server contains DDE items that can be continually updated, data can be received and updated automatically by creating *TDdeClientItem* objects for those items.

The value of the DDE item depends on the linked DDE server application. *Item* is typically a selectable portion of text, such as a spreadsheet cell or a database field in an edit box. If the DDE server is an Delphi application, *Item* is the name of the linked DDE server component.

Note See the documentation for the DDE server application for specific information about specifying a DDE item.

See also *StrDispose* function, *TDdeClientItem* object

SetLink

SetLink specifies the service and topic of a DDE conversation and attempts to open the link if *ConnectMode* is *ddeAutomatic*.

```
function SetLink(Service: string; Topic: string): Boolean;
```

Description Call *SetLink* to fully specify a DDE conversation. If *ConnectMode* is *ddeManual*, calling *SetLink* is equivalent to setting the *DdeTopic* and *DdeService* properties, and clearing all data from any associated *TDdeClientItem* objects. If *ConnectMode* is *ddeAutomatic*, *SetLink* opens the link to the DDE server as well.

The *Service* and Topic parameters specify the values for the *DdeTopic* and *DdeService* properties, respectively.

See also *ConnectMode* property, *DdeService* property, *DdeTopic* property, *OpenLink* method

StartAdvise

StartAdvise starts an advise loop for all of the T*DdeClientItems* associated with the current DDE conversation.

```
function StartAdvise: Boolean;
```

Description Call *StartAdvise* to start an advise loop for all of the T*DdeClientItems* associated with the current DDE conversation. *StartAdvise* returns *True* if loop startup is successful.

See also *TDdeClientItem* object

TDdeClientItem

Unit: DDEMan | Palette: System

TObject • TPersistent • TComponent • TDdeClientItem

TDdeClientItem defines an item in a DDE conversation.

Description Use *TDdeClientItem* with *TDdeClientConv* to enable an application to act as a DDE client. A DDE client item specifies a single item supported by a DDE server. A single *TDdeClientConv* object may be associated with multiple client items. If the DDE client conversation has established a link with a DDE server, the server will automatically and continually update all client items until the conversation is terminated.

See also *TDdeClientConv* object, *TDdeServerItem* object

Properties

The following table lists the properties of *TDdeClientItem*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DdeConv	published		TDdeClientItem
DdeItem	published		TDdeClientItem
DesignInfo	public		TComponent
Lines	published		TDdeClientItem
Name	published		TComponent
Owner	public, read-only		TComponent
Tag	published	0	TComponent
Text	published		TDdeClientItem
VCLComObject	public		TComponent

DDEConv

DDEConv specifies the DDE client conversation to associate with the DDE client item.

```
property DdeConv: TDDEClientConv;
```

Description Set *DDEConv* to associate the DDE client item with a DDE client conversation. The value of *DdeConv* is the DDE client conversation object that

defines the DDE conversation. The *DdeConv* establishes the link to the server that provides the *Text* for the DDE client item.

See also *Lines* property, *TDDeClientConv* object, *Text* property

DDEItem

DDEItem specifies the item of a DDE conversation.

```
property DDEItem: string;
```

Description Set the *DDEItem* property to the name of a DDE item supported by the DDE server. The value of *DDEItem* depends on the linked DDE server application. *DDEItem* is typically a selectable portion of text, such as a spreadsheet cell or a database field in an edit box. If the DDE server is a Delphi application, *DDEItem* is the name of the linked DDE server item.

At design time, specify *DDEItem* by typing the item string in the Object Inspector. Alternately, *DDEItem* can be specified from the DDE conversation by pasting a link using the DDE Info dialog box, which is called by clicking the ellipsis (...) button for *DdeService* or *DdeTopic* in the Object Inspector. After choosing Paste Link in the DDE Info dialog box, choose the item from a list of possible items for *DDEItem* in the Object Inspector when link information is on the Clipboard.

Note See the documentation for the DDE server application for the specific information about specifying DDE items.

See also *TDdeClientConv.DdeService* property, *TDdeClientConv.DdeTopic* property, *TDdeServerItem* object

Lines

Lines contains the text data to exchange in a DDE conversation.

```
property Lines: TStrings;
```

Description Read *Lines* to get the text that is updated by the DDE server application. *Lines* is updated automatically by the DDE server. When *Lines* is changed by the server, an *OnChange* event occurs.

Set the *Lines* property to poke data to the DDE server.

Lines corresponds to the *Text* property. Whenever the value of *Lines* or *Text* is updated, the other is also changed so that the first line of *Lines* is always equal to *Text*.

Note Lines of data can also be sent to the DDE server by calling the *PokeDataLines* method of the DDE client conversation.

See also *TDdeClientConv.PokeDataLines* method, *Text* property, *TStrings* object

Text

Text contains the text data to exchange in a DDE conversation.

```
property Text: string;
```

Read *Text* to get the text that is updated by the DDE server application. *Text* is updated automatically by the DDE server. When *Text* is changed by the server, an *OnChange* event occurs.

Set the *Text* property to poke data to the DDE server.

Text corresponds to the *Lines* property. Whenever the value of *Lines* or *Text* is updated, the other is also changed so that the first line of *Lines* is always equal to *Text*.

For text data longer than the 255 character limit of *Text*, use the *Lines* property.

Note Data can also be sent to the DDE server by calling the *PokeData* method of the DDE client conversation.

See also *Lines* property, *TDdeClientConv.PokeData* method

Events

The following table lists the events of *TDdeClientItem*:

Event	Visibility	Documented in
OnChange	published	TDdeClientItem

OnChange

OnChange occurs immediately after the DDE server updates the *Lines* property.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnChange: TNotifyEvent;
```

Description Write an *OnChange* event handler to take specific action when the DDE server updates the text for this DDE client item. The new value for the DDE client item can be found by reading the *Lines* or *Text* property.

See also *Lines* property, *Text* property

Methods

The following table lists the methods of *TDdeClientItem*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TDdeClientItem
DefaultHandler	public	TObject
Destroy	public	TDdeClientItem
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Create

Create creates an instance of *TDdeClientItem*.

constructor Create(AOwner: TComponent); **override**;

Call *Create* to instantiate a *TDdeClientItem* at runtime. After calling the inherited constructor, *Create* creates a helper object to manage the *Lines* property.

See also *Lines* property, *TComponent.Create* method

Destroy

Destroy destroys an instance of *TDdeClientItem*.

destructor Destroy; **override**;

Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the *TDdeClientItem* is not already freed, and only then calls *Destroy*.

Destroy frees the memory associated with the *Lines* property before calling the inherited destructor.

See also *Free* method, *Lines* property

TDdeServerConv

Unit: DDEMan | Palette: System

TObject • TPersistent • TComponent • TDdeServerConv

TDdeServerConv represents a DDE topic of conversation with a DDE client.

Description Use *TDdeServerConv* to establish a DDE topic with a DDE client application. Individual items under the DDE topic are represented by *TDdeServerItem* objects. These items are linked to the DDE server conversation by setting their *ServerConv* property.

Using a *TDDEServerConv* component is optional. Without a *TDDEServerConv* object, the client can still enter into a DDE conversation and receive an update directly from a *TDDEServerItem* object. However, a DDE server conversation is required to handle any macros that the DDE client may send.

The *Name* property of the DDE server conversation provides the topic used by DDE clients to establish a link to the DDE server conversation. DDE server items that are not associated with a DDE server conversation use the *Caption* property of the form for the topic name. Use a *TDDEServerConv* object when establishing multiple DDE topics in a single form, or when the *Caption* of the form may not be unique or constant at runtime.

See also *TDdeClientConv* object, *TDdeCientItem* object, *TDdeServerItem* object, *TForm* object

Properties

The following table lists the properties of *TDdeServerConv*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DesignInfo	public		TComponent
Name	published		TComponent
Owner	public, read-only		TComponent
Tag	published	0	TComponent
VCLComObject	public		TComponent

Events

The following table lists the events of *TDdeServerConv*:

Event	Visibility	Documented in
OnClose	published	TDdeServerConv
OnExecuteMacro	published	TDdeServerConv
OnOpen	published	TDdeServerConv

OnClose

OnClose occurs when the DDE client terminates the DDE conversation.

```
type TNotifyEvent = procedure(Sender: TObject) of object;
property OnClose: TNotifyEvent;
```

Description Write an *OnClose* event handler to take specific action when the DDE conversation ends. The DDE conversation will end if the client terminates the conversation. It does not end if the *TDdeServerConv* object is destroyed. In that case, the DDE server items simply remove themselves from the DDE server conversation, and continue the link to the DDE client.

The *Sender* parameter is the DDE server conversation, not the DDE client.

See also *Destroy* method, *OnOpen* event

OnExecuteMacro

OnExecuteMacro occurs when the DDE client sends a macro to the DDE server conversation.

```
type TMacroEvent = procedure(Sender: TObject; Msg: TStrings) of object;
property OnExecuteMacro: TMacroEvent;
```

Description Write an *OnExecuteMacro* event handler to implement a response to macros sent by the DDE client. The *Sender* parameter is the DDE server conversation. The *Msg* parameter gives the macro string to execute.

See also *ExecuteMacro* method

OnOpen

OnOpen occurs immediately after the DDE link to the client has been established.

```
type TNotifyEvent = procedure(Sender: TObject) of object;
property OnOpen: TNotifyEvent;
```

Description Write an *OnOpen* event handler to take specific action whenever a link is established to a DDE client. The *Sender* parameter is the DDE server conversation, not the DDE client.

See also *OnClose* event

Methods

The following table lists the methods of *TDdeServerConv*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TDdeServerConv
DefaultHandler	public	TObject
Destroy	public	TDdeServerConv
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
ExecuteMacro	public	TDdeServerConv

Method	Visibility	Documented in
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Create

Create creates an instance of *TDdeServerConv*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to create an instance of *TDdeServerConv* at runtime. For *TDdeServerConv* objects placed on forms at design time, *Create* is called automatically.

See also *TComponent.Create* method

Destroy

Destroy destroys an instance of *TDdeServerConv*.

destructor Destroy; **override**;

Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the *TDdeServerConv* object is not already freed and only then calls *Destroy*.

Destroy removes the DDE server conversation topic from the application, but does not terminate any currently open DDE conversations. Instead, the DDE

server items that were associated with the DDE server conversation become associated with the form instead, and continue their link to the DDE client.

See also *Free* method, *TDdeServerItem* object

ExecuteMacro

ExecuteMacro responds to macros sent by the DDE client.

```
function ExecuteMacro(Data: HDdeData): LongInt;
```

Description *ExecuteMacro* provides the only way for a Delphi application to respond to macros sent by DDE clients. *ExecuteMacro* is called when the DDE client sends a Macro over the DDE link. *Data* is the handle to the DDE data that contains the macro.

ExecuteMacro retrieves the strings of the macro from *Data*, and calls the *OnExecuteMacro* event handler. If there is no *OnExecuteMacro* event handler, the macro is ignored. Whether or not there is an *OnExecuteMacro* event handler, the DDE client always receives notification that the macro was successfully transmitted, unless *Data* contains no macro.

See also *OnExecuteMacro* event

TDdeServerItem

Unit: DDEMan | Palette: System

TObject • TPersistent • TComponent • TDdeServerItem

TDDEServerItem defines the item of a DDE conversation.

Description Use *TDDEServerItem* by itself, or, optionally, with a *TDDEServerConv* object to enable an application to function as a DDE server. DDE clients link to an individual DDE server item by referring to the *Name* property. Once a link to a DDE client is established, the DDE server item can update the client to changes in the item data, or receive data that the DDE client pokes to the item.

See also *TDdeClientConv* object, *TDdeClientItem* object, *TDdeServerConv* object

Properties

The following table lists the properties of *TDdeServerItem*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent

Property	Visibility	Default	Documented in
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DesignInfo	public		TComponent
Fmt	public, read-only		TDdeServerItem
Lines	published		TDdeServerItem
Name	published		TComponent
Owner	public, read-only		TComponent
ServerConv	published		TDdeServerItem
Tag	published	0	TComponent
Text	published		TDdeServerItem
VCLComObject	public		TComponent

Fmt

Fmt is the clipboard format the DDE client item uses to pass data to the DDE client.

property Fmt: Integer;

Description Set *Fmt* in an *OnChange* event handler to specify a clipboard format for the data that is passed to the DDE client if it is not text. The DDE data that is sent to the DDE client includes this format so that clients can correctly interpret the data. Read *Fmt* when the client pokes data to the DDE server item to determine the clipboard format the client used to format the data.

See also *Lines* property, *OnChange* event, *OnPokeData* event, *TDdeClientConv.DdeFmt* property, *Text* property

Lines

Lines contains the data to exchange in a DDE conversation.

property Lines: TStrings;

Description Set *Lines* to the data the DDE server item sends to the DDE client. Whenever the *Lines* property changes, an *OnChange* event occurs and then the new value is sent to the DDE client. If the DDE server item accepts poked data, *Lines* receives the data whenever the DDE client pokes data to this item. When *Lines* is changed by a DDE client poking data, an *OnPokeData* event occurs after the *OnChange* event.

Usually *Lines* contains text, and setting *Lines* sets the *Fmt* property to CF_TEXT. However, Delphi strings can contain non-text data. If *Lines* does not specify text data, *Fmt* must be changed to indicate the format of the data in an *OnChange* event handler.

Lines corresponds to the *Text* property. Whenever the value of *Lines* or *Text* is changed, the other is updated so that the first line of *Lines* is always equal to *Text*.

See also *Fmt* property, *OnChange* event, *OnPokeData* event, *Text* property, *TStrings* object

ServerConv

ServerConv specifies the DDE server conversation to associate with the DDE server item.

```
property ServerConv: TDdeServerConv;
```

Description Set *ServerConv* to associate the DDE server item with a DDE topic represented by the DDE server conversation. If *ServerConv* is not set, the DDE topic of the conversation is the value of the *Caption* property of the form containing the *TDDEServerItem* component.

See also *TDdeServerConv* object

Text

Text contains the data to exchange in a DDE conversation.

```
property Text: string;
```

Description Set *Text* to the data the DDE server item sends to the DDE client. Whenever the *Text* property changes, an *OnChange* event occurs and then the new value is sent to the DDE client. If the DDE server item accepts poked data, *Text* receives the data whenever the DDE client pokes data to this item. When *Text* is changed by a DDE client poking data, an *OnPokeData* event occurs after the *OnChange* event.

Usually *Text* contains text, and setting *Text* sets the *Fmt* property to CF_TEXT. However, Delphi strings can contain non-text data. If *Text* does not specify text data, *Fmt* must be changed to indicate the format of the data in an *OnChange* event handler.

Text corresponds to the *Lines* property. Whenever the value of *Lines* or *Text* is changed, the other is updated so that the first line of *Lines* is always equal to *Text*.

If the data to be exchanged is greater than 255 characters, use *Lines* instead.

See also *Fmt* property, *Lines* property, *OnChange* event, *OnPokeData* event

Events

The following table lists the events of *TDdeServerItem*:

Event	Visibility	Documented in
OnChange	published	TDdeServerItem
OnPokeData	published	TDdeServerItem

OnChange

OnChange occurs when the *Lines* property changes, before the changes have been sent to the DDE client.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnChange: TNotifyEvent;
```

Description Write an *OnChange* event handler to take specific action when the DDE server item is about to update the DDE client. The new value for the DDE client item can be found by reading the *Lines* or *Text* property.

If the value of *Lines* does not contain text data, an *OnChange* event handler must be assigned to change the value of *Fmt*. *OnChange* is called after *Fmt* gets set to CF_TEXT by setting the *Lines* property, but before the data gets sent to the client.

See also *Change* method, *Fmt* property, *Lines* property, *Text* property

OnPokeData

OnPokeData occurs after a DDE client changes the value of the *Lines* property.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnPokeData: TNotifyEvent;
```

Description Write an *OnPokeData* event handler to take specific action when the DDE client pokes data to this DDE server item. The new value for the item can be found by reading the *Lines* or *Text* property.

When a DDE client pokes data to a DDE server items, the *PokeData* method is called. *PokeData* causes the following events.

1. First, the new value is assigned to the *Lines* (*Text*) property.
2. An *OnChange* event occurs.
3. The new value is sent back to the DDE client.
4. An *OnPokeData* event occurs.

See also *Lines* property, *OnChange* event, *PokeData* method, *Text* property

Methods

The following table lists the methods of *TDdeServerItem*:

Method	Visibility	Documented in
Assign	public	TPersistent
Change	public	TDdeServerItem
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
CopyToClipboard	public	TDdeServerItem
Create	public	TDdeServerItem
DefaultHandler	public	TObject
Destroy	public	TDdeServerItem
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PokeData	public	TDdeServerItem
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Change

Change causes an *OnChange* event.

procedure Change; **dynamic**;

Description Call *Change* to trigger an *OnChange* event when the *Lines* or *Text* property are not set. Use *Change* in conjunction with the *OnChange* event to enable the DDE server item to exchange data that is not compatible with the strings used by the *Lines* and *Text* properties.

See also *Lines* property, *OnChange* event, *Text* property

CopyToClipboard

CopyToClipboard copies the data specified in the *Text* or *Lines* property to the Windows Clipboard, along with DDE link information.

procedure CopyToClipboard;

Description Use *CopyToClipboard* to enable DDE clients to link to the DDE server item by executing an Edit | Paste Link command, or its equivalent in the command structure of the DDE client application.

CopyToClipboard can be used to create a DDE link at runtime only. To create a link at design time, select the *TDdeServerItem* object and choose Edit | Copy from the menu. Then, activate the DDE client application and paste the link according to the rules of the DDE client application. See the documentation for the DDE client application for specific information about pasting the link.

See also *Lines* property, *TDdeClientConv.PasteLink* method, *Text* property

Create

Create creates an instance of *TDdeServerItem*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to instantiate a DDE server item at runtime. *Create* is called automatically for *TDdeServerItem* objects placed on forms at design time.

After calling the inherited constructor, *Create* initializes the *Fmt* property to CF_TEXT, and creates a helper object to manage the *Lines* property.

See also *Fmt* property, *Lines* property, *TComponent.Create* method

Destroy

Destroy destroys an instance of *TDdeServerItem*.

destructor Destroy; **override**;

Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the *TDdeServerItem* is not already freed, and only then calls *Destroy*.

Destroy frees the memory associated with the *Lines* property before calling the inherited destructor.

See also *Lines* property

PokeData

PokeData processes data that has been poked to the DDE server item by a DDE client.

```
function PokeData(Data: HDdeData): LongInt;
```

Description Call *PokeData* to inform the DDE server item that the DDE conversation holds data that has been poked from a DDE client. *Data* is the DDE handle to the poked data.

PokeData does the following:

1. Uses the *Data* handle to obtain the data that was poked.
2. Sets the *Lines* property to the new data value.
3. Triggers an *OnChange* event.
4. Sends the new value of the data back to the DDE client.
5. Triggers an *OnPokeData* event.

See also *Lines* property, *OnChange* event, *OnPokeData* event

TDecisionCube

Unit: MXDB | Palette: Decision Cube

TObject • TPersistent • TComponent • TCustomDataStore • TDecisionCube

TDecisionCube maintains crosstabulated data that it provides to a *TDecisionSource* object.

Description Add a *TDecisionCube* object to a form or data module to allow an application to represent crosstabulated data. *TDecisionCube* analyzes the data from a source dataset into a multidimensional structure where each dimension corresponds to a field in the dataset over which information is summarized. Any dataset can be used as the source of a *TDecisionCube* object, but it is easiest to work with *TDecisionQuery*.

TDecisionCube introduces properties and methods that allow a crosstabulated data store to provide its data to a *TDecisionSource* object. Decision grids and Decision graphs do not retrieve data directly from the Decision cube. They use a *TDecisionSource* object which represents a configuration of the dimensions from the *TDecisionCube* object.

See also *TDecisionGraph* object, *TDecisionGrid* object, *TDecisionPivot* object, *TDecisionQuery* object, *TDecisionSource* object

Properties

The following table lists the properties of *TDecisionCube*:

Property	Visibility	Default	Documented in
Active	public		TDecisionCube
BinData	public, read-only		TCustomDataStore
Capacity	public		TCustomDataStore
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
CurrentSummary	public		TCustomDataStore
DataSet	published		TCustomDataStore
DesignInfo	public		TComponent
DesignState	public		TCustomDataStore
DimensionCount	public, read-only		TCustomDataStore
DimensionMap	published		TCustomDataStore
DimensionMapCount	public, read-only		TCustomDataStore
MaxDimensions	published		TCustomDataStore
MaxSummaries	published		TCustomDataStore
Name	published		TComponent
Owner	public, read-only		TComponent
ShowProgressDialog	published		TCustomDataStore
SummaryCount	public, read-only		TCustomDataStore
Tag	published	0	TComponent
VCLComObject	public		TComponent

Active

Active specifies whether the Decision cube can provide data.

```
property Active: Boolean;
```

Description Read *Active* to determine whether the Decision cube is ready to provide data to a Decision control. Set *Active* to *True* to cause the Decision cube to fetch and analyze data from the dataset specified in the *DataSet* property.

When *Active* is *False*, the Decision cube is closed and the Decision source can't read data from the Decision cube. When *Active* is *True*, data can be read from the Decision cube.

Note The *Active* property cannot be set to *True* unless the dataset specified by the *DataSet* property is already active.

See also *DataSet* property, *AfterClose* event, *AfterOpen* event *BeforeClose* event, *BeforeOpen* event

Events

The following table lists the event of *TDecisionCube*:

Event	Visibility	Documented in
AfterClose	published	TCustomDataStore
AfterOpen	published	TCustomDataStore
BeforeClose	published	TCustomDataStore
BeforeOpen	published	TCustomDataStore
OnLowCapacity	published	TCustomDataStore
OnRefresh	published	TCustomDataStore

Methods

The following table lists the methods of *TDecisionCube*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TDecisionCube
DefaultHandler	public	TObject
Destroy	public	TDecisionCube
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetDetailSQL	public	TDecisionCube
GetDimensionMemberCount	public	TCustomDataStore
GetDimensionName	public	TCustomDataStore
GetInterface	public	TObject

Method	Visibility	Documented in
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetMemberAsString	public	TCustomDataStore
GetMemberAsVariant	public	TCustomDataStore
GetMemoryUsage	public	TCustomDataStore
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetSQL	public	TDecisionCube
GetSummaryAsString	public	TCustomDataStore
GetSummaryAsVariant	public	TCustomDataStore
GetSummaryName	public	TCustomDataStore
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Refresh	public	TCustomDataStore
RemoveComponent	public	TComponent
SafeCallException	public	TComponent
ShowCubeDialog	public	TDecisionCube

Create

Create creates an instance of *TDecisionCube*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to instantiate a Decision cube at runtime. Decision cubes placed on a form or data module at design time are created automatically.

Create allocates the memory for the Decision cube and its helper objects and initializes the *State* property to *dcInactive* and the *DesignState* property to *dsAllData*.

See also *DesignState* property, *State* property, *TCustomDataStore.Create* method

Destroy

Destroy destroys an instance of *TDecisionCube*.

destructor Destroy; **override**;

Description Do not call the *Destroy* method directly. Instead call *Free*. *Free* verifies that the memory for the Decision cube is not already freed, and only then calls *Destroy*.

See also *Free* method

GetDetailSQL

GetDetailSQL returns an SQL statement that can be used to generate the described subset of the data in the Decision cube.

```
function GetDetailSQL( ValueArray: TSmallIntArray; SelectList: string; bActive: Boolean): string;
```

Description Use the SQL statement returned by *GetDetailSQL* to generate a dataset that represents the records that gave rise to a subset of the values in the Decision cube. The resulting query produces the records represented in the projection described by the *ValueArray* parameter. The dataset will include all fields that make up the dimensions of the Decision cube, plus any fields listed in the *SelectList* parameter. The query does not include any summary values or grouping, such as must be imposed by the dataset of the Decision cube.

Each entry in the *ValueArray* parameter corresponds to a dimension of the Decision cube, where the first entry corresponds to the first dimension, the second entry corresponds to the second dimension, and so on. If the value of an entry in the value array is less than 0, all field values over that dimension are included in the new dataset. If the value of an entry in the value array is equal to or greater than 0, it is interpreted as the index of a value over the dimension field. When the *ValueArray* specifies a field value, the SQL statement restricts the dataset to records that match that value on the corresponding field of the original database table. Thus, *ValueArray* represents the dimension array with a set of drilled dimensions.

bActive specifies whether the *ValueArray* parameter includes inactive dimensions. Only active dimensions are included when *bActive* is *True*. Also when *bActive* is *True*, only the fields that correspond to active dimensions are included automatically in the SQL query.

The *SelectList* parameter is a list of comma delimited field names for the fields that are included in the result set of the SQL query. The query always includes the fields that correspond to the dimensions represented by the *ValueArray* parameter. If *SelectList* consists of a single asterisk (*), all the fields in the original dataset are included.

See also *DataSet* property, *GetSQL* method, *TQuery.SQL* property, *TSmallIntArray* object

GetSQL

GetSQL returns an SQL statement that can be used to generate the described subset of the data in the Decision cube

```
function GetSQL( ValueArray: TSmallIntArray; bActive: Boolean): string;
```

Description Use the SQL statement returned by *GetSQL* to view the data that gave rise to a subset of the information in the Decision cube. The resulting query produces a dataset containing the projection described by the *ValueArray* parameter. All the summaries of the current dataset are included in the resulting query.

Each entry in the *ValueArray* parameter corresponds to a dimension of the Decision cube, where the first entry corresponds to the first dimension, the second entry corresponds to the second dimension, and so on. If the value of an entry in the value array is less than 0, all field values over that dimension are included in the new dataset. If the value of an entry in the value array is equal to or greater than 0, it is interpreted as the index of a value over the dimension field. When the *ValueArray* specifies a field value, the SQL statement restricts the dataset to records that match that value on the corresponding field of the original database table. Thus, *ValueArray* represents the dimension array with a set of drilled dimensions.

bActive specifies whether the *ValueArray* parameter includes inactive dimensions. Only active dimensions are included when *bActive* is *True*.

To further restrict the dataset so that it includes only a subset of the dimensions in the Decision cube, use the *GetDetailSQL* method.

See also *DataSet* property, *GetDetailSQL* method, *TQuery.SQL* property, *TSmallIntArray* object

ShowCubeDialog

ShowCubeDialog displays the Decision cube editor.

```
procedure ShowCubeDialog;
```

Description Call *ShowCubeDialog* to allow the user to change the active dimensions of the data cube using the Decision cube editor. *ShowCubeDialog* brings up the same dialog that appears when Decision cube editor is selected from the pop-up menu of the Decision cube component at design time.

The Decision cube editor allows the user to make dimensions active or inactive and to change the names of dimensions. It also controls how much information is shown in a Decision control at design time.

See also *DimensionCount* property, *DimensionMap* property, *SummaryCount* property, *GetDimensionName* method

TDecisionGraph

Unit: MXGraph | Palette: Decision Cube

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomControl • TCustomPanel • TCustomTeePanel • TCustomChart • TChart • TCustomDecisionGraph • TDecisionGraph

TDecisionGraph displays a graph of crosstabulated data.

Description

Use *TDecisionGraph* to put a Decision graph in a form. Decision graphs chart a single summary that is calculated over a number of dimensions. The axes of the graph correspond to these dimensions, which are provided by a Decision source object. By default, the first row dimension appears as the x-axis and the first column dimension appears as the y-axis.

The field that is summarized and the fields (dimensions) over which it is summarized can be changed dynamically at runtime to view any configuration of summary and dimensions chosen from the dimension map of a Decision cube. Changes to the dimensions and summary can be initiated using a *TDecisionPivot* object.

TDecisionGraph publishes many of the properties, events, and methods inherited from *TCustomDecisionGraph*, but introduces no new behavior.

To present the crosstabulated data in a grid, use *TDecisionGrid* instead.

See also

TDecisionCube object, *TDecisionGrid* object, *TDecisionPivot* object, *TDecisionSource* object

Properties

The following table lists the properties of *TDecisionGraph*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
AllowPanning	published	pmBoth	TChart
AllowZoom	published	True	TChart
AnimatedZoom	published	False	TChart
AnimatedZoomSteps	published	8	TChart
AxisVisible	published	True	TChart (TCustomChart)
BackColor	published	clTeeColor	TChart (TCustomChart)
BackImage	published		TChart
BackImageInside	published	False	TChart
BackImageMode	published	pbmStretch	TChart
BevelInner	published	bvNone	TCustomPanel
BevelOuter	published	bvRaised	TCustomPanel
BevelWidth	published	1	TCustomPanel
BorderStyle	published	bsNone	TCustomPanel

Property	Visibility	Default	Documented in
BorderWidth	published	0	TCustomPanel
BottomAxis	published		TChart (TCustomChart)
BottomWall	published		TChart
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
BufferedDisplay	public		TChart (TCustomChart)
Canvas	public, read-only		TChart (TCustomControl)
Chart3DPercent	published	Def3DPercent	TChart (TCustomChart)
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ClipPoints	published	True	TChart (TCustomChart)
Color	published	clBtnFace	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DecisionSource	published		TCustomDecisionGraph
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Foot	published		TChart
Frame	published		TChart (TCustomChart)
Gradient	published		TChart
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
LeftAxis	published		TChart (TCustomChart)
LeftWall	published		TChart
Legend	published		TChart (TCustomChart)

Property	Visibility	Default	Documented in
Locked	published	False	TCustomPanel
MarginBottom	published	TeeDefVerticalMargin	TChart (TCustomTeePanel)
MarginLeft	published	TeeDefHorizMargin	TChart (TCustomTeePanel)
MarginRight	published	TeeDefHorizMargin	TChart (TCustomTeePanel)
MarginTop	published	TeeDefVerticalMargin	TChart (TCustomTeePanel)
MaxPointsPerPage	published	0	TChart (TCustomChart)
Monochrome	published	False	TChart (TCustomChart)
MoveOptions	published		TCustomDecisionGraph
Name	published		TControl
Owner	public, read-only		TComponent
Page	published	1	TChart (TCustomChart)
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
PrintResolution	public	0	TChart
RightAxis	published		TChart (TCustomChart)
SaveSeries	published		TCustomDecisionGraph
ScaleLastPage	published	True	TChart (TCustomChart)
Series	public, read-only		TChart (TCustomChart)
SeriesList	published, read-only		TChart (TCustomChart)
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	False	TWinControl
Tag	published	0	TComponent
Title	published		TChart
Top	published		TControl
TopAxis	published		TChart (TCustomChart)
VCLComObject	public		TComponent
View3d	published	True	TChart (TCustomChart)
View3dWalls	published	True	TChart (TCustomChart)
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl
Zoomed	public		TChart

Events

The following table lists the events of *TDecisionGraph*

Event	Visibility	Documented in
OnAfterDraw	published	TChart (TCustomChart)
OnAllowScroll	published	TChart
OnClick	published	TControl
OnClickAxis	published	TChart
OnClickBackground	published	TChart
OnClickLegend	published	TChart
OnClickSeries	published	TChart
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnGetAxisLabel	published	TChart (TCustomChart)
OnGetLegendPos	published	TChart
OnGetLegendRect	published	TChart
OnGetLegendText	published	TChart (TCustomChart)
OnGetNextAxisLabel	published	TChart (TCustomChart)
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnPageChange	published	TChart (TCustomChart)
OnResize	published	TCustomPanel
OnScroll	published	TChart
OnUndoZoom	published	TChart
OnZoom	published	TChart

Methods

The following table lists the methods of *TDecisionGraph*:

Method	Visibility	Documented in
ActiveSeriesLegend	public	undocumented
AddSeries	public	TChart (TCustomChart)
Assign	public	TPersistent
BackWallRect	public	undocumented
BeginDrag	public	TControl
BringToFront	public	TControl

Method	Visibility	Documented in
Broadcast	public	TWinControl
CalcClickedPart	public	undocumented
CalcMetaBounds	public	undocumented
CalcSize3d	public	TChart (TCustomChart)
CalcSize3dWalls	public	TChart (TCustomChart)
CanFocus	public	TWinControl
CanvasChanged	public	undocumented
ChartPrintRect	public	TChart
ChartRegionRect	public	TChart (TCustomChart)
CheckDatasource	public	TChart (TCustomChart)
CheckPenWidth	public	undocumented
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ClipRectangle	public	undocumented
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
CopyToClipboardBitmap	public	undocumented
CopyToClipboardMetafile	public	undocumented
Create	public	TCustomDecisionGraph
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TCustomDecisionGraph
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DoHorizLine	public	undocumented
DoLine	public	undocumented
DoPolygon	public	undocumented
DoRectangle	public	TChart (TCustomTeePanel)
DoVertLine	public	undocumented
DragDrop	public	TControl
Dragging	public	TControl
Draw	public	TChart (TCustomChart)
DrawToMetaCanvas	public	undocumented
EnableAlign	public	TWinControl
EndDrag	public	TControl

Method	Visibility	Documented in
ExchangeSeries	public	undocumented
FieldAddress	public	TObject
FillSeriesSourceItems	public	undocumented
FillValueSourceItems	public	undocumented
FindComponent	public	TComponent
Focused	public	TWinControl
FontCanvas	public	TChart (TCustomTeePanel)
FormattedLegend	public	TChart (TCustomChart)
FormattedValueLegend	public	TChart (TCustomChart)
Free	public	TObject
FreeAllSeries	public	undocumented
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetASeries	public	TChart (TCustomChart)
GetAxisSeries	public	TChart (TCustomChart)
GetCursorPos	public	TChart (TCustomTeePanel)
GetEditorClass	public	TChart
GetFreeSeriesColor	public	TChart (TCustomChart)
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetLabelsSeries	public	TChart (TCustomChart)
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetRectangle	public	undocumented
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
GetWidthHeight	public	undocumented
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
InternalDraw	public	undocumented
Invalidate	public	TWinControl
IsFreeSeriesColor	public	TChart (TCustomChart)

Method	Visibility	Documented in
IsScreenHighColor	public	TChart
IsValidDataSource	public	TChart (TCustomChart)
MarkText	public	TChart (TCustomChart)
MaxMarkWidth	public	TChart (TCustomChart)
MaxTextWidth	public	TChart (TCustomChart)
MaxXValue	public	TChart (TCustomChart)
MaxYValue	public	TChart (TCustomChart)
MethodAddress	public	TObject
MethodName	public	TObject
MinXValue	public	TChart (TCustomChart)
MinYValue	public	TChart (TCustomChart)
NewInstance	public	TObject
NextPage	public	TChart
NumPages	public	TChart (TCustomChart)
PaintTo	public	TWinControl
Perform	public	TControl
PreviousPage	public	TChart
Print	public	TChart
PrintLandscape	public	TChart
PrintOrientation	public	TChart
PrintPartial	public	TChart
PrintPartialCanvas	public	TChart
PrintPortrait	public	TChart
PrintRect	public	TChart
Realign	public	TWinControl
ReCalcWidthHeight	public	TChart (TCustomTeePanel)
Refresh	public	TControl
RemoveAllSeries	public	TChart (TCustomChart)
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
RemoveSeries	public	TChart (TCustomChart)
Repaint	public	TWinControl
RotateLabel	public	TChart (TCustomTeePanel)
SafeCallException	public	TComponent
SaveToBitmapFile	public	TChart
SaveToMetafile	public	TChart
SaveToMetafileEnh	public	undocumented
SaveToMetafileRect	public	undocumented
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl

Method	Visibility	Documented in
SeriesCount	public	TChart (TCustomChart)
SeriesDown	public	TChart
SeriesTitleLegend	public	undocumented
SeriesUp	public	TChart
SetBooleanProperty	public	undocumented
SetBounds	public	TWinControl
SetBrushCanvas	public	TChart (TCustomTeePanel)
SetColorProperty	public	undocumented
SetDoubleProperty	public	undocumented
SetFocus	public	TWinControl
SetIntegerProperty	public	undocumented
SetInternalCanvas	public	undocumented
SetLongintProperty	public	undocumented
SetStringProperty	public	undocumented
SetTextBuf	public	TControl
Show	public	TControl
TeeCharHeight	public	undocumented
TeeCreateMetafile	public	undocumented
TeeTextHeight	public	TChart (TCustomTeePanel)
TeeTextOut	public	TChart (TCustomTeePanel)
TeeTextWidth	public	TChart (TCustomTeePanel)
UnClipRectangle	public	undocumented
UndoZoom	public	TChart
Update	public	TWinControl
UpdateControlState	public	TWinControl
XLabelText	public	TChart (TCustomChart)
ZoomPercent	public	TChart
ZoomRect	public	TChart

TDecisionGrid

Unit: MXGrid

Palette: Decision Cube

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomControl • TCustomGrid • TCustomDecisionGrid • TDecisionGrid

TDecisionGrid presents crosstabulated data from a Decision source in a grid format.

Description

Use *TDecisionGrid* to put a Decision grid in a form. Decision grids present a single summary that is calculated over an arbitrary number of dimensions. The dimensions can appear as rows or columns in the grid, with the field values of the dimensions listed as labels along the top and left edges of the grid.

The field that is summarized and the fields (dimensions) over which it is summarized can be changed dynamically at runtime to view any configuration of summary and dimensions chosen from the dimension map of a Decision cube. Changes to the dimensions and summary can be initiated by direct manipulation of the Decision grid, or by using a *TDecisionPivot* object.

TDecisionGrid publishes many of the properties, events, and methods inherited from *TCustomDecisionGrid*, but introduces no new behavior.

To present the crosstabulated data in a chart, use *TDecisionGraph* instead.

See also *TDecisionCube* object, *TDecisionGraph* object, *TDecisionPivot* object, *TDecisionSource* object

Properties

The following table lists the properties of *TDecisionGrid*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
BorderStyle	published	bsSingle	TCustomGrid
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
CaptionColor	published		TCustomDecisionGrid
CaptionFont	published		TCustomDecisionGrid
Cells	public, read-only		TCustomDecisionGrid
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ColCount	public, read-only	5	TCustomDecisionGrid
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DataColor	published		TCustomDecisionGrid

Property	Visibility	Default	Documented in
DataFont	published		TCustomDecisionGrid
DataSumColor	published		TCustomDecisionGrid
DecisionSource	published		TCustomDecisionGrid
DefaultColWidth	published	ColWidth	TCustomDecisionGrid
DefaultDrawing	published	Drawing	TCustomGrid
DefaultRowHeight	published	RowHeight	TCustomDecisionGrid
DesignInfo	public		TComponent
Dimensions	published		TCustomDecisionGrid
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
FixedCols	public, read-only	1	TCustomDecisionGrid
FixedRows	public, read-only	1	TCustomDecisionGrid
GridLineColor	published		TCustomDecisionGrid
GridLineWidth	published	1	TCustomDecisionGrid
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
LabelColor	published		TCustomDecisionGrid
LabelFont	published		TCustomDecisionGrid
LabelSumColor	published		TCustomDecisionGrid
Left	published		TControl
Name	published		TControl
Options	published	[cgGridLines, cgOutLiner, cgPivotable]	TCustomDecisionGrid
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
RowCount	public, read-only	5	TCustomDecisionGrid
ScrollBars	published	ssBoth	TCustomGrid
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent

Property	Visibility	Default	Documented in
Top	published		TControl
Totals	public		TCustomDecisionGrid
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Events

The following table lists the events of *TDecisionGrid*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDblClick	published	TControl
OnDecisionDrawCell	published	TCustomDecisionGrid
OnDecisionExamineCell	published	TCustomDecisionGrid
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnSelectCell	published	TCustomDecisionGrid
OnStartDrag	published	TControl
OnTopLeftChanged	published	TCustomDecisionGrid

Methods

The following table lists the methods contained in *TDecisionGrid*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl

Method	Visibility	Documented in
CellDrawState	public	TCustomDecisionGrid
CellValueArray	public	TCustomDecisionGrid
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomDecisionGrid
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TCustomDecisionGrid
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl

Method	Visibility	Documented in
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
MouseCoord	public	TCustomGrid
NewInstance	public	TObject
Notification	public	TCustomDecisionGrid
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

TDecisionPivot

Unit: MXPivSrc

Palette: Decision Cube

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomControl • TCustomPanel • TDecisionPivot

TDecisionPivot allows the user to control the pivot state of a Decision source.

Description

Add a Decision pivot to a form to allow users to control the pivot state of Decision graphs and Decision grids. Decision pivots contain buttons that correspond to the active dimensions of a Decision cube and a drop-down list box for choosing the current summary.

Pressing the dimension buttons opens or closes the dimensions, controlling whether the Decision source provides detailed data values for those dimensions. Right-clicking the buttons of a Decision pivot brings up a menu that allows users to control the drill state of a dimension. Drilled dimensions focus on a single value within the dimension. Dragging the buttons on a Decision pivot rearranges the dimensions of the Decision source into a new pivot configuration.

See also *TDecisionCube* object, *TDecisionGraph* object, *TDecisionGrid* object, *TDecisionSource* object

Properties

The following table lists the properties of *TDecisionPivot*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
Alignment	published	taCenter	TCustomPanel
BevelInner	published	bvNone	TCustomPanel
BevelOuter	published	bvRaised	TCustomPanel
BevelWidth	published	1	TCustomPanel
BorderStyle	published	bsNone	TDecisionPivot
BorderWidth	published	0	TDecisionPivot
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ButtonAutoSize	published		TDecisionPivot
ButtonHeight	published	30	TDecisionPivot
ButtonSpacing	published		TDecisionPivot
ButtonWidth	published	60	TDecisionPivot
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clBtnFace	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl

Property	Visibility	Default	Documented in
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DecisionSource	published		TDecisionPivot
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Font	published		TControl
GroupLayout	published	xtLeftTop	TDecisionPivot
Groups	published		TDecisionPivot
GroupSpacing	published		TDecisionPivot
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

BorderStyle

BorderStyle determines whether a line is drawn around the perimeter of the Decision pivot.

property BorderStyle: TBorderStyle;

Description Use *BorderStyle* to specify whether the Decision pivot has a single line drawn around it. When *BorderStyle* is *bsSingle*, a single line is drawn around the panel outside of the outer bevel. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

Don't confuse the line drawn around the Decision pivot with the *BorderWidth* property. The *BorderWidth* is the distance between the outer and inner bevels.

See also *BorderWidth* property

BorderWidth

BorderWidth is the distance, in pixels, between the outer and inner bevels.

```
type TBorderWidth: 0..MaxInt;
```

```
property BorderWidth: TBorderWidth;
```

Description Use *BorderWidth* to specify how wide the border around the Decision pivot should be. A value of 0 means no border should appear.

The border of the Decision pivot is the area between the outer and inner bevels. It is visible only if the inner bevel is raised or lowered, but affects the inset of the controls in the pivot even if *BevelInner* is *bvNone*.

Don't confuse the border of the pivot with line drawn around it. The line around the pivot is specified by the *BorderStyle* property.

See also *BorderStyle* property, *BevelInner* property, *BevelOuter* property

ButtonAutoSize

ButtonAutoSize specifies whether the dimension buttons resize automatically when the Decision pivot resizes.

```
property ButtonAutoSize: Boolean;
```

Description Set *ButtonAutoSize* to *True* to cause the dimension buttons to expand and contract automatically when the size of the Decision pivot changes. Set *ButtonAutoSize* to *False* to specify fixed size buttons.

Note The value of *ButtonAutoSize* is ignored when *GroupLayout* is *xtTopLeft*.

See also *ButtonHeight* property, *ButtonWidth* property, *GroupLayout* property

ButtonHeight

ButtonHeight is the height, in pixels, of the dimension buttons.

```
property ButtonHeight: Integer;
```

Description Use *ButtonHeight* to determine the *Height* of the dimension buttons. Do not set *ButtonHeight* when the *ButtonAutoSize* property is *True*. When *ButtonAutoSize* is *True*, the height of the buttons is reset automatically whenever the layout of the Decision pivot changes.

See also *ButtonAutoSize* property, *ButtonWidth* property, *NewPanelSetup* method

ButtonSpacing

ButtonSpacing is the number of pixels that appear between dimension buttons.

```
property ButtonSpacing: Integer;
```

Description Set *ButtonSpacing* to control the spacing between the buttons in a single group. Increase *ButtonSpacing* to make a Decision pivot look less crowded. Decrease *ButtonSpacing* to make the row and column groups look more compact.

See also *GroupSpacing* property

ButtonWidth

ButtonWidth is the width, in pixels, of the dimension buttons.

```
property ButtonWidth: Integer;
```

Description Use *ButtonWidth* to determine the *Width* of the dimension buttons. Do not set *ButtonWidth* when the *ButtonAutoSize* property is *True*. When *ButtonAutoSize* is *True*, the width of the buttons is reset automatically whenever the layout of the Decision pivot changes.

See also *ButtonAutoSize* property, *ButtonHeight* property, *NewPanelSetup* method

DecisionSource

DecisionSource is the Decision source object that the Decision pivot controls.

```
property DecisionSource: TDecisionSource;
```

Description Set *DecisionSource* to link the Decision pivot to a Decision source component. The controls in the Decision pivot change the pivot state of the Decision source.

The pivot state of the Decision source can also change in response to another control, such as a Decision grid. When the pivot state of the Decision source changes, the Decision pivot updates the layout of the dimension buttons and the selection in the summary combo box.

See also *NewPanelSetup* method, *TDecisionGrid* object, *TDecsionSource* object

GroupLayout

GroupLayout specifies how the dimension buttons and summary control are arranged on the Decision pivot.

```
type TDecisionButtonPosition = (xtHorizontal, xtVertical, xtLeftTop);

property GroupLayout: TDecisionButtonPosition;
```

Description Set *Layout* to specify the organization of a Decision pivot. *Layout* can have one of the following values:

Value	Meaning
xtHorizontal	The controls in the pivot are lined up side by side.
xtVertical	The controls in the pivot are lined up top to bottom.
xtLeftTop	The controls in the pivot are lined up along the edges of the pivot, with the row dimensions down the left side, the column dimensions along the top, and the summary control in the upper left corner.

When *Layout* is *xtLeftTop*, the center of the Decision pivot is empty. A Decision graph or Decision grid can be placed in this empty spot so that the buttons line up with the dimensions they control.

When *Layout* is *xtHorizontal* or *xtVertical*, the summary control appears first, followed by a the row dimensions and then the column dimensions. The *Groups* property can omit any of these from the layout.

See also *Groups* property

Groups

Groups specifies which controls appear in a Decision pivot.

```
type
  TDecisionPivotOption = (xtRows, xtColumns, xtSummaries);
  TDecisionPivotOptions = set of TDecisionPivotOption;

property Groups: TDecisionPivotOptions;
```

Description Set *Groups* to limit the types of controls that appear in the Decision pivot. This allows greater flexibility over where the controls appear in a form. For example, a form can contain three Decision pivots: one containing only row dimension buttons, one containing only column dimension buttons, and one containing only the summary selection combo box. These three pivots can then be arranged anywhere on the form.

Groups is a set containing one or more of the following values:

Value	Meaning
xtRows	The pivot contains row dimension buttons.

Value	Meaning
xtColumns	The pivot contains column dimension buttons.
xtSummaries	The pivot contains a summary selection combo box.

See also *GroupLayout* property

GroupSpacing

GroupSpacing is the number of pixels that appear between control groups.

property GroupSpacing: Integer;

Description Set *GroupSpacing* to control the spaces that appear between the summary combo box, the group of row dimension buttons, and the group of column dimension buttons. Increase *GroupSpacing* to make the groups of controls appear more distinct. Decrease *GroupSpacing* to make the Decision pivot look more compact.

See also *ButtonSpacing* property, *Groups* property

Events

The following table lists the events for *TDecisionPivot*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnResize	published	TCustomPanel
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TDecisionPivot*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl

Method	Visibility	Documented in
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TDecisionPivot
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TDecisionPivot
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject

Method	Visibility	Documented in
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TDecisionPivot
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

Create

Create creates and initializes an instance of *TDecisionPivot*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to instantiate a Decision pivot at runtime. Decision pivots placed on forms at design time are created automatically. After calling the inherited constructor, *Create* initializes the following properties:

- *BorderStyle* to *bsNone* and *BorderWidth* to 3.
- *ButtonAutoSize* to *True*.
- *ButtonHeight* to 24 and *ButtonWidth* to 64.
- *ButtonSpacing* to 3 and *GroupSpacing* to 10.
- *Groups* to [*xtRows, xtColumns, xtSummaries*].
- *GroupLayout* to *xtHorizontal*.

See also *BorderStyle* property, *BorderWidth* property, *ButtonAutoSize* property, *ButtonHeight* property, *ButtonSpacing* property, *ButtonWidth* property, *GroupLayout* property, *Groups* property, *GroupSpacing* property, *TCustomPanel.Create* method

Destroy

Destroy destroys an instance of *TDecsionPivot*.

```
destructor Destroy; override;
```

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the Decision pivot is not already freed, and only then calls *Destroy*.

See also *Free* method.

SetBounds

SetBounds resizes the Decision pivot.

```
procedure SetBounds(Left, Top, Height, Width: Integer); override;
```

Description *Use SetBounds* to change all of the Decision pivot's boundary properties at once. The same effect can be achieved by setting the *Left*, *Top*, *Width*, and *Height* properties separately. By setting all four properties at once, *SetBounds* ensures that the control will not repaint between changes. Specify the values for the *Left*, *Top*, *Width*, and *Height* properties as the value of the *Left*, *Top*, *Width*, and *Height* parameters, respectively.

SetBounds overrides the inherited method to adjust the layout of the buttons in the Decision pivot when the size changes.

See also *ButtonAutoSize* property, *Height* property, *Left* property, *NewPanelSetup* method, *Top* property, *Width* property

TDecisionQuery

Unit: MXTables Palette: Decision Cube

TObject • TPersistent • TComponent • TDataSet • TBDEDataSet • TDBDataSet • TQuery • TDecisionQuery

TDecisionQuery is a specialized query that prepares crosstabulated data for a Decision cube.

Description Use a *TDecisionQuery* as the dataset of a Decision cube to ensure that the data is properly prepared. *TDecisionQuery* allows the dimensions and summaries to be defined directly on the dataset analyzed in a Decision control. It then prepares an SQL query to provide the information from that dataset in a format that can be used by a Decision cube.

Decision cubes can use other types of datasets, but the information must be structured in the same way as the output of a *TDecisionQuery* object.

See also *TDecisionCube* object

Properties

The following table lists the properties of *TDecisionQuery*:

Property	Visibility	Default	Documented in
Active	published	False	TDataSet
AutoCalcFields	published	True	TDataSet
BOF	public, read-only		TDataSet
Bookmark	public		TDataSet
CacheBlobs	public	True	TBDEDataSet
CachedUpdates	published	False	TBDEDataSet
CanModify	public, read-only		TBDEDataSet
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Constrained	published	False	TQuery
Constraints	published		TDataSet
Database	public, read-only		TDBDataSet
DatabaseName	published		TDBDataSet
DataSource	published		TQuery
DBHandle	public, read-only		TDBDataSet
DBLocale	public, read-only		TDBDataSet
DBSession	public, read-only		TDBDataSet
DefaultFields	public, read-only		TDataSet
Designer	public, read-only		TDataSet
DesignInfo	public		TComponent
EOF	public, read-only		TDataSet
ExpIndex	public, read-only		TBDEDataSet
FieldCount	public, read-only		TDataSet
FieldDefs	public		TDataSet
Fields	public		TDataSet
FieldValues	public		TDataSet
Filter	published		TBDEDataSet
Filtered	published	False	TBDEDataSet
FilterOptions	published	[]	TBDEDataSet
Found	public, read-only		TDataSet

Property	Visibility	Default	Documented in
Handle	public, read-only		TBDEDataSet
KeySize	public, read-only		TBDEDataSet
Local	public, read-only		TQuery
Locale	public, read-only		TBDEDataSet
Modified	public, read-only		TDataSet
Name	published		TDataSet
Owner	public, read-only		TComponent
ParamCheck	published	True	TQuery
ParamCount	public, read-only		TQuery
Params	published		TQuery
Prepared	public		TQuery
Provider	public, read-only		TDBDataSet
RecNo	public		TBDEDataSet
RecordCount	public, read-only		TBDEDataSet
RecordSize	public, read-only		TBDEDataSet
RequestLive	published	False	TQuery
RowsAffected	public, read-only		TQuery
SessionName	published		TDBDataSet
SQL	published		TQuery
SQLBinary	public		TQuery
State	public, read-only		TDataSet
StmtHandle	public, read-only		TQuery
Tag	published	0	TComponent
Text	public, read-only		TQuery
UniDirectional	published	False	TQuery
UpdateMode	published	upWhereAll	TDBDataSet
UpdateObject	published		TBDEDataSet
UpdateRecordTypes	public		TBDEDataSet
UpdatesPending	public, read-only		TBDEDataSet
VCLComObject	public		TComponent

Events

The following table lists the events for *TDecisionQuery*:

Event	Visibility	Documented in
AfterCancel	published	TDataSet
AfterClose	published	TDataSet
AfterDelete	published	TDataSet
AfterEdit	published	TDataSet
AfterInsert	published	TDataSet

Event	Visibility	Documented in
AfterOpen	published	TDataSet
AfterPost	published	TDataSet
AfterScroll	published	TDataSet
BeforeCancel	published	TDataSet
BeforeClose	published	TDataSet
BeforeDelete	published	TDataSet
BeforeEdit	published	TDataSet
BeforeInsert	published	TDataSet
BeforeOpen	published	TDataSet
BeforePost	published	TDataSet
BeforeScroll	published	TDataSet
OnCalcFields	published	TDataSet
OnDeleteError	published	TDataSet
OnEditError	published	TDataSet
OnFilterRecord	published	TDataSet
OnNewRecord	published	TDataSet
OnPostError	published	TDataSet
OnServerYield	published	TBDEDataSet
OnUpdateError	published	TBDEDataSet
OnUpdateRecord	published	TBDEDataSet

Methods

The following table lists the methods of *TDecisionQuery*:

Method	Visibility	Documented in
ActiveBuffer	public	TDataSet
Append	public	TDataSet
AppendRecord	public	TDataSet
ApplyUpdates	public	TBDEDataSet
Assign	public	TPersistent
BookmarkValid	public	TBDEDataSet
Cancel	public	TBDEDataSet
CancelUpdates	public	TBDEDataSet
CheckBrowseMode	public	TDataSet
CheckOpen	public	TDBDataSet
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject

Method	Visibility	Documented in
CleanupInstance	public	TObject
ClearFields	public	TDataSet
Close	public	TDataSet
CommitUpdates	public	TBDEDataSet
CompareBookmarks	public	TBDEDataSet
ConstraintCallBack	public	TBDEDataSet
ControlsDisabled	public	TDataSet
Create	public	TQuery
CursorPosChanged	public	TDataSet
DefaultHandler	public	TObject
Delete	public	TDataSet
Destroy	public	TQuery
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableConstraints	public	TBDEDataSet
DisableControls	public	TDataSet
Dispatch	public	TObject
Edit	public	TDataSet
EnableConstraints	public	TBDEDataSet
EnableControls	public	TDataSet
ExecSQL	public	TQuery
FetchAll	public	TBDEDataSet
FieldAddress	public	TObject
FieldByName	public	TDataSet
FindComponent	public	TComponent
FindField	public	TDataSet
FindFirst	public	TDataSet
FindLast	public	TDataSet
FindNext	public	TDataSet
FindPrior	public	TDataSet
First	public	TDataSet
Free	public	TObject
FreeBookmark	public	TDataSet
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetBookmark	public	TDataSet
GetCurrentRecord	public	TBDEDataSet
GetFieldList	public	TDataSet
GetFieldNames	public	TDataSet
GetIndexInfo	public	TBDEDataSet
GetInterface	public	TObject

Method	Visibility	Documented in
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GotoBookmark	public	TDataSet
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
Insert	public	TDataSet
InsertComponent	public	TComponent
InsertRecord	public	TDataSet
InstanceSize	public	TObject
IsEmpty	public	TDataSet
IsLinkedTo	public	TDataSet
Last	public	TDataSet
Locate	public	TBDEDataSet
Lookup	public	TBDEDataSet
MethodAddress	public	TObject
MethodName	public	TObject
MoveBy	public	TDataSet
NewInstance	public	TObject
Next	public	TDataSet
Open	public	TDataSet
ParamByName	public	TQuery
Post	public	TDataSet
Prepare	public	TQuery
Prior	public	TDataSet
Refresh	public	TDataSet
RemoveComponent	public	TComponent
Resync	public	TDataSet
RevertRecord	public	TBDEDataSet
SafeCallException	public	TComponent
SetFields	public	TDataSet
Translate	public	TDataSet
UnPrepare	public	TQuery
UpdateCursorPos	public	TDataSet
UpdateRecord	public	TDataSet
UpdateStatus	public	TBDEDataSet

TDecisionSource

Unit: MXDB | Palette: Decision Cube

TObject • TPersistent • TComponent • TDecisionSource

TDecisionSource links a Decision control to a configuration (pivot state) of the data from a Decision cube.

Description

Use *TDecisionSource* to provide a Decision grid or Decision graph with cross-tabulated data from a Decision cube. *TDecisionSource* represents a particular pivot state of dimensions from the Decision cube into row and column dimensions. *TDecisionSource* also provides methods to drill down the display of a dimension to focus on summaries over a single value, or hide or show dimensions in the pivot state.

All Decision controls that share a Decision source display the same pivot state. For example, a form could contain a Decision grid and a Decision graph linked to the same Decision source. Pivoting the dimensions of the Decision grid would automatically update the image of the graph accordingly. To display two separate projections of the information from a Decision cube in a single form, use two separate Decision sources.

The pivot state of a Decision source can be changed by manipulating a Decision grid or by using a Decision pivot.

See also

TDecisionCube object, *TDecisionGraph* object, *TDecisionGrid* object, *TDecisionPivot* object

Properties

The following table lists the properties of *TDecisionSource*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlType	published		TDecisionSource
CurrentSum	public, read-only		TDecisionSource
DecisionCube	published		TDecisionSource
DesignInfo	public		TComponent
Name	published		TComponent
nColDims	public, read-only		TDecisionSource
nDataCols	public, read-only		TDecisionSource
nDataRows	public, read-only		TDecisionSource

Property	Visibility	Default	Documented in
nDims	public, read-only		TDecisionSource
nOpenColDims	public, read-only		TDecisionSource
nOpenRowDims	public, read-only		TDecisionSource
nRowDims	public, read-only		TDecisionSource
nSums	public, read-only		TDecisionSource
Owner	public, read-only		TComponent
Ready	public, read-only		TDecisionSource
SparseCols	published		TDecisionSource
SparseRows	published		TDecisionSource
Tag	published	0	TComponent
VCLComObject	public		TComponent

ControlType

ControlType specifies how the Decision source responds to messages from the Decision pivot.

```
type TDecisionControlType = (xtCheck, xtRadio, xtRadioEx);
```

```
property ControlType: TDecisionControlType;
```

Description Use *ControlType* to specify what combination of row or column dimensions can be open at the same time. *ControlType* has one of the following values:

Value	Meaning
xtCheck	Pressing dimension buttons opens or closes an individual dimension, changing the number of open dimensions.
xtRadio	Pressing dimension buttons opens or closes the selected dimension and closes all other dimensions in that direction. When *ControlType* is *xtRadio*, at most one row dimension and one column dimension may be open at a time.
xtRadionEx	Pressing dimension buttons opens the selected dimension and closes all other dimensions in that direction or closes the selected dimension and opens another dimension in the same direction. When *ControlType* is *xtRadioEx*, exactly one row dimension and one column dimension must be open at any time.

See also *Dimensions* property, *TDecisionPivot* object

CurrentSum

CurrentSum is the index of the current summary from the set of active summaries maintained by the Decision cube.

```
property CurrentSum: integer;
```

Description Read *CurrentSum* to determine which of the active summaries is used to provide values to clients of the Decision source. *CurrentSum* returns 0 if the

first active summary is the current summary. *CurrentSum* returns *nSums*–1 if the last active summary is the current summary.

To change the current summary, use the *SetCurrentSummary* method. Use the *GetSummaryName* method to obtain the name of the current summary.

See also *GetSummaryName* method, *nSums* property, *OnSummaryChange* event, *SetCurrentSummary* method

DecisionCube

DecisionCube is the Decision cube that provides the crosstabulated data for the Decision source.

```
property DecisionCube: TDecisionCube;
```

Description Set *DecisionCube* to specify where the Decision source should fetch its data. The Decision cube maintains a multidimensional array of summaries in a cache. The Decision source can provide Decision controls with any projection of the data in that cache.

See also *TDecisionCube* object

nColDims

nColDims is the number of column dimensions in the current pivot state of the Decision source.

```
property nColDims: Integer;
```

Description Read *nColDims* to determine the number of column dimensions the Decision source can provide to a Decision control. Column dimensions appear on the y-axis of a Decision graph or listed along the top of a Decision grid. *nColDims* counts both the open and closed column dimensions. To determine the number of column dimensions actually provided to Decision controls, use the *nOpenColDims* property.

See also *GetActiveDim* method, *nDims* property, *nOpenColDims* property, *nRowDims* property

nDataCols

nDataCols is the number of columns provided to the data region of a Decision grid.

```
property nDataCols: Integer;
```

Description Read *nDataCols* to determine the number of summarized values the Decision source provides for a row in a Decision grid. Each of the columns counted by *nDataCols* represents a combination of values or dimension subtotals for each of the open column dimensions.

See also *nOpenColDims* property

nDataRows

nDataRows is the number of rows provided to the data region of a Decision grid.

```
property nDataRows: Integer;
```

Description Read *nDataRows* to determine the number of summarized values the Decision source provides for a column in a Decision grid. Each of the rows counted by *nDataRows* represents a combination of values or dimension subtotals for each of the open row dimensions.

See also *nOpenRowDims* property

nDims

nDims is the number of active dimensions in the Decision cube.

```
property nDims: Integer;
```

Description Read *nDims* to determine the number of dimensions the Decision source can provide to a Decision control. The Decision source does not necessarily provide Decision controls with all the dimensions counted by the *nDims* property. Some of the dimensions may be closed.

The dimensions available through the Decision source are divided into column dimensions and row dimensions. These are counted by the *nColDims* and *nRowDims* properties, respectively. *nDims* is equal to the sum of *nColDims* and *nRowDims*.

See also *nColDims* property, *nRowDims* property

nOpenColDims

nOpenColDims is the number of column dimensions provided by the Decision source.

```
property nOpenColDims: Integer;
```

Description Read *nOpenColDims* to determine the number of open column dimensions the Decision source provides to a Decision control. Column dimensions appear on the y-axis of a Decision graph or listed along the top of a Decision grid. *nOpenColDims* counts only the open column dimensions that appear in the Decision control. To determine the number of column dimensions potentially available in the current pivot state, use the *nColDims* property.

See also *GetActiveDim* method, *nColDims* property, *nOpenRowDims* property

nOpenRowDims

nOpenRowDims is the number of row dimensions provided by the Decision source.

property nOpenRowDims: Integer;

Description Read *nOpenRowDims* to determine the number of open row dimensions the Decision source provides to a Decision control. Row dimensions appear on the x-axis of a Decision graph or listed along the left side of a Decision grid. *nOpenRowDims* counts only the open row dimensions that appear in the Decision control. To determine the number of row dimensions potentially available in the current pivot state, use the *nRowDims* property.

See also *GetActiveDim* method, *nOpenColDims* property, *nRowDims* property

nRowDims

nRowDims is the number of row dimensions in the current pivot state of the Decision source.

property nRowDims: Integer;

Description Read *nRowDims* to determine the number of row dimensions the Decision source can provide to a Decision control. Row dimensions appear on the x-axis of a Decision graph or listed along the left side of a Decision grid. *nRowDims* counts both the open and closed row dimensions. To determine the number of row dimensions actually provided to Decision controls, use the *nOpenRowDims* property.

See also *GetActiveDim* method, *nColDims* property, *nDims* property, *nOpenRowDims* property

nSums

nSums is the number of active summaries in the Decision cube.

property nSums: Integer;

Description Read *nSums* to determine the number of summary types the Decision source can provide to a Decision control. The Decision source can only provide Decision controls with a single summary at a time. The *CurrentSum* property indicates which of the *nSums* possibilities is currently provided by the Decision source.

See also *CurrentSum* property, *SetCurrentSummary* method

Ready

Ready indicates whether the Decision source is linked to an active Decision cube.

property Ready: Boolean;

Description Read *Ready* to determine whether the Decision source can provide data to a Decision control.

See also *DecisionCube* property, *TDecisionCube.Active* property

SparseCols

SparseCols indicates whether empty columns are removed from the current projection.

```
property SparseCols: boolean;
```

Description Set *SparseCols* to *True* to display columns that are empty in the current pivot state. Set *SparseCols* to *False* to show only those dimension values that have summarized data over one or more of the row dimensions.

Note *SparseCols* doesn't change the value returned by *GetDimensionMemberCount*.

See also *GetDimensionMemberCount* method, *SparseRows* method

SparseRows

SparseRows indicates whether empty rows are removed from the current projection.

```
property SparseRows: boolean;
```

Description Set *SparseRows* to *True* to display rows that are empty in the current pivot state. Set *SparseRows* to *False* to show only those dimension values that have summarized data over one or more of the column dimensions.

Note *SparseRows* doesn't change the value returned by *GetDimensionMemberCount*.

See also *GetDimensionMemberCount* method, *SparseCols* method

Events

The following table lists the events of *TDecisionSource*:

Event	Visibility	Documented in
OnBeginPivot	published	TDecisionSource
OnEndPivot	published	TDecisionSource
OnLayoutChange	published	TDecisionSource
OnNewDimensions	published	TDecisionSource
OnStateChange	published	TDecisionSource
OnSummaryChange	published	TDecisionSource

OnBeginPivot

OnBeginPivot occurs just before the current pivot state changes.

```
type TNotifyEvent = procedure(Sender: TObject) of object;
property OnBeginPivot: TNotifyEvent;
```

Description Write an *OnBeginPivot* event handler to take specific action just before the information provided by the Decision source changes. When *OnPivot* occurs, the results of the pivot have been computed but not implemented. Use the *GetGroup*, *GetIndex*, *GetState*, *GetValue*, and *GetRowState* methods to get information about the pivot state that is about to be implemented. To get information about the pivot state that is ending, save the dimension information in an *OnEndPivot* event handler. The information about the new state can then be compared to information captured the last time the pivot state was changed. The *OnBeginPivot* event handler can make additional alterations to the pivot state using the methods of the Decision source.

See also *OnEndPivot* event, *GetGroup* method, *GetIndex* method, *GetRowState* method, *GetState* method, *GetValue* method

OnEndPivot

OnEndPivot occurs immediately after the current pivot state changes.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnEndPivot: TNotifyEvent;
```

Description Write an *OnEndPivot* event handler to take specific action after the information provided by the Decision source changes but before any Decision controls are updated to reflect the change. This is a good opportunity to save information about the pivot state.

When the pivot state of the Decision source changes, an *OnBeginPivot* occurs when the results of the change have been calculated, and *OnEndPivot* occurs when the Decision source has been updated to reflect the change. Once the Decision controls that are linked to the Decision pivot are updated to reflect the change, an *OnLayoutChange* event occurs.

See also *OnBeginPivot* event, *OnLayoutChange* event, *OnNewDimensions* event, *OnStateChange* event

OnLayoutChange

OnLayoutChange occurs immediately after Decision controls are updated to reflect a change to the pivot state of the Decision source.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnLayoutChange: TNotifyEvent;
```

Description Write an *OnLayoutChange* event handler to respond to changes in the configuration of the active dimensions. *OnLayoutChange* occurs when the pivot or drill state changes, but the information cached by the Decision cube remains the same. When *OnLayoutChange* occurs, the information the Decision source can potentially provide does not change, only the current configuration of that information. To respond to changes in the data cached by the Decision cube, use the *OnNewDimensions* event.

See also *OnEndPivot* event, *OnNewDimensions* event

OnNewDimensions

OnNewDimensions occurs immediately after the data provided by the Decision cube changes.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnNewDimensions: TNotifyEvent;
```

Description Write an *OnNewDimensions* event handler to respond to changes that alter the data array, such as when the active property of a dimension in the data cube changes. When *OnNewDimensions* occurs, the information the Decision source can provide changes. To respond to changes that reorganize the data but do not change the data cached by the data cube, use the *OnEndPivot* or *OnLayoutChange* event. To respond to changes in the data cube itself, use the *OnStateChange* event.

See also *Dimensions* property, *OnEndPivot* event, *OnLayoutChange* event, *OnStateChange* event, *OnSummaryChange* event

OnStateChange

OnStateChange occurs when the *DecisionCube* property changes.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnStateChange: TNotifyEvent;
```

Description Write an *OnStateChange* event handler to respond when the *DecisionCube* property changes or when the Decision cube's *Active* property changes. To respond to less drastic changes, use the *OnNewDimensions* event, the *OnLayoutChange* event, or the *OnEndPivot* event.

See also *DecisionCube* property, *OnEndPivot* event, *OnLayoutChange* event, *OnNewDimensions* event, *TDecisionCube.Active* property

OnSummaryChange

OnSummaryChange occurs immediately after the CurrentSum property changes.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnSummaryChange: TNotifyEvent;
```

Description Write an *OnSummaryChange* event handler to take specific action when the summary provided by the Decision source changes. The value of the *CurrentSum* property reflects the new current summary.

See also *CurrentSum* property, *OnNewDimensions* event

Methods

The following table lists the methods of *TDecisionSource*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
CloseDimIndexRight	public	TDecisionSource
Create	public	TDecisionSource
DefaultHandler	public	TObject
Destroy	public	TDecisionSource
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
DrillDimIndex	public	TDecisionSource
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
Get2DDataAsVariant	public	TDecisionSource
GetActiveDim	public	TDecisionSource
GetDataAsString	public	TDecisionSource
GetDataAsVariant	public	TDecisionSource
GetDimensionMemberCount	public	TDecisionSource
GetDimensionName	public	TDecisionSource
GetGroup	public	TDecisionSource
GetGroupExtent	public	TDecisionSource
GetIndex	public	TDecisionSource
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetMemberAsString	public	TDecisionSource
GetMemberAsVariant	public	TDecisionSource
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetRowState	public	TDecisionSource

Method	Visibility	Documented in
GetState	public	TDecisionSource
GetSummaryName	public	TDecisionSource
GetValue	public	TDecisionSource
GetValueArray	public	TDecisionSource
GetValueIndex	public	TDecisionSource
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
MoveDimIndexes	public	TDecisionSource
NewInstance	public	TObject
OpenDimIndexLeft	public	TDecisionSource
OpenDimIndexRight	public	TDecisionSource
RemoveComponent	public	TComponent
SafeCallException	public	TComponent
SetCurrentSummary	public	TDecisionSource
SwapDimIndexes	public	TDecisionSource
ToggleDimIndex	public	TDecisionSource

CloseDimIndexRight

CloseDimIndexRight closes a dimension.

```
type TDimGroup = (dgRow, dgCol, dgSum, dgPage, dgNone);
procedure CloseDimIndexRight(dimGroup: TDimGroup; Index: Integer; bOpen: boolean);
```

Description Call *CloseDimIndexRight* to close a dimension so that it does not appear in the projection displayed by Decision controls that use the Decision source.

CloseDimIndexRight closes the first open dimension after the dimension identified by the *Index* parameter. If the *dimGroup* parameter is *dgRow*, *Index* identifies a row dimension, where 0 specifies the first row dimension, 1 specifies the second row dimension, and so on. If the *dimGroup* parameter is *dgCol*, *Index* identifies a column dimension, starting with an index of 0 for the first column dimension. No other values for *dimGrouop* should be used with the *CloseDimIndexRight* method. To close the first open row or column dimension, set *Index* to –1. The *bOpen* parameter specifies whether closed dimensions are counted when interpreting *Index*. Only open dimensions are counted when *bOpen* is *True*.

See also *GetActiveDim* method, *GetRowState* method, *OpenDimIndexLeft* method, *OpenDimIndexRight* method, *ToggleDimIndex* method

Create

Create creates and initializes an instance of *TDecisionSource*.

```
constructor Create( AOwner: TComponent ); override;
```

Description Call *Create* to instantiate a Decision source at runtime. Decision source objects placed in forms or data modules at design time are created automatically.

See also *TComponent.Create* method

Destroy

Destroy destroys an instance of *TDecisionSource*.

```
destructor Destroy; override;
```

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the Decision source is not already freed and only then calls *Destroy*.

For all Decision controls linked to the Decision source, *Destroy* changes the *DecisionSource* property to **nil**.

See also *Free* method, *TCustomDecisionGraph.DecisionSource* property, *TCustomDecisionGrid.DecisionSource* property

DrillDimIndex

DrillDimIndex collapses a dimension to focus on a single value.

```
type TDimGroup = (dgRow, dgCol, dgSum, dgPage, dgNone);

procedure DrillDimIndex(dimGroup: TDimGroup; Index: Integer; ValueIndex: Integer;
  bOpen:boolean);
```

Description Call *DrillDimIndex* to "drill down" on a dimension, so that only the summaries for a single value are displayed.

The *dimGroup* parameter specifies the type of dimension to be drilled. If *dimGroup* is *dgRow*, the *Index* parameter identifies a row dimension to be drilled. If *dimGroup* is *dgCol*, *Index* identifies a column dimension to be drilled. No other values for *dimGroup* should be used with the *DrillDimIndex* method. The first row or column dimension has an *Index* of 0, the second dimension has an *Index* of 1, and so on. The *ValueIndex* parameter is the index of single value from all values over the dimension that should be displayed. To display only the summaries over the entire dimension, rather than for a single value along the dimension, set *ValueIndex* to –1. The *bOpen* parameter specifies whether closed dimensions are counted when interpreting *Index*. Only open dimensions are counted when *bOpen* is *True*.

To restore the dimension from its drilled state so that it displays all values again, use the *ToggleDimIndex* method.

See also *GetActiveDim* method, *GetMemberAsString* method, *GetValueIndex* method, *ToggleDimIndex* method

Get2DDataAsVariant

Get2DDataAsVariant returns the summary for a two-dimensional projection of the current pivot state.

```
function Get2DDataAsVariant(iDimA, iDimB: Integer; aValueIndex, bValueIndex:Integer):
  Variant;
```

Description Call *Get2DDataAsVariant* to obtain the summary value from the Decision cube that corresponds to the coordinates of *aValueIndex* on the *iDimA* dimension and *bValueIndex* on the *iDimB* dimension. The returned value is a variant that summarizes all values over the other active dimensions, unless they are drilled. For drilled dimensions, only the drilled value is included in the summary.

The *iDimA* and *iDimB* parameters specify the dimensions for the two dimensional projection, where 0 indicates the first active dimension, 1 indicates the second active dimension, and so on. The dimension indexes are expressed as absolute indexes. To obtain an absolute index from a row or column index, use the *GetActiveDim* method. The *aValueIndex* and *bValueIndex* parameters are the indexes of the values on the two dimensions, where 0 indicates the first value, 1 indicates the second value, and so on. Use the *GetMemberAsString* or *GetMemberAsVariant* method to obtain the field values that correspond to the *aValueIndex* and *bValueIndex* parameters.

See also *Dimensions* property, *DrillDimIndex* method, *GetActiveDim* method, *GetDataAsVariant* method, *GetMemberAsString* method, *GetMemberAsVariant* method

GetActiveDim

GetActiveDim converts a row-specific or column-specific dimension index into an absolute dimension index.

```
type TDimGroup = (dgRow, dgCol, dgSum, dgPage, dgNone);

function GetActiveDim(dimGroup: TDimGroup; Index: Integer; bOpen: boolean): Integer;
```

Description Call *GetActiveDim* to obtain the index of a dimension in the Decision cube from its position in a Decision grid. The names and values of row dimensions appear in the label cells on the right of the Decision grid. The first row dimension appears in the first column, the second row dimension appears in the second column, and so on. Similarly, column dimensions appear in the label cells on the top of the Decision grid, with the first column dimension appearing in the first row of the grid.

The *dimGroup* parameter indicates whether the dimension of interest is a row dimension or column dimension. If *dimGroup* is *dgRow*, the dimension is a row dimension. If *dimGroup* is *dgCol*, the dimension is a column dimension.

GetActiveDim is not implemented for other values of *dimGroup*. The *Index* parameter indicates the position of the dimension in the set of row or column dimensions, where 0 specifies the first row or column dimension, 1 specifies the second dimension, and so on. The *bOpen* parameter specifies whether closed dimensions are counted when interpreting *Index*. Only open dimensions are counted when *bOpen* is *True*.

GetActiveDim returns the absolute index of the dimension. Absolute indexes are used by many methods of the Decision source, such as *GetDimensionName* or *GetMemberAsString*.

Note To obtain a row-specific index or column-specific index from an absolute dimension index, use the *GetIndex* method.

See also *Dimensions* property, *GetDimensionName* method, *GetGroup* method, *GetIndex* method, *GetMemberAsString* method

GetDataAsString

GetDataAsString returns the data value that should appear in the cell of a Decision grid, formatted as a string.

```
function GetDataAsString(ARow, ACol:Integer; var SubLevel: Integer): String;
```

Description Call *GetDataAsString* to obtain the display string for a data cell in a Decision grid. The *ARow* and *ACol* parameters indicate the row and column numbers of the cell in the Decision grid, where (0,0) specifies the top left cell in the data region.

GetDataAsString computes the dimension coordinates in the Decision cube that correspond to the row and column indexes, making note of all dimensions over which subtotals must be taken. It then obtains the summary value from the Decision cube, combining values for each summarized dimension. *GetDataAsString* returns the final summary value, formatted as a string. It also returns the number of summarized dimensions in the *SubLevel* parameter.

See also *GetDataAsVariant* method

GetDataAsVariant

GetDataAsVariant returns the data value that should appear in the cell of a Decision grid.

```
function GetDataAsVariant(Arow, ACol: Integer; var SubLevel: Integer): Variant;
```

Description Call *GetDataAsVariant* to obtain the value of a data cell in a Decision grid. The *ARow* and *ACol* parameters indicate the row and column numbers of the cell in the Decision grid, where (0,0) specifies the top left cell in the data region.

GetDataAsVariant computes the dimension coordinates in the Decision cube that correspond to the row and column indexes, making note of all

dimensions over which subtotals must be taken. It then obtains the summary value from the Decision cube, combining values for each summarized dimension. *GetDataAsVariant* returns the final summary value. It also returns the number of summarized dimensions in the *SubLevel* parameter.

See also *Get2DDataAsVaraint* method, *GetDataAsString* method

GetDimensionMemberCount

GetDimensionMemberCount returns the number of distinct values to be displayed for a given dimension.

```
function GetDimensionMemberCount(iDim: Integer): Integer;
```

Description Call *GetDimensionMemberCount* to determine the number of values in the dimension identified by the *iDim* parameter. The *iDim* parameter is the absolute index of the dimension, where 0 indicates the first active dimension in the Decision cube, 1 indicates the second active dimension, and so on. To convert from a row-specific or column-specific dimension index into an absolute dimension index, use the *GetActiveDim* method.

GetDimensionMemberCount provides an upper limit to the *ValueIndex* parameter of the *GetMemberAsString* or *GetMemberAsVariant* method.

Note The value returned by *GetDimensionMemberCount* may change depending on the pivot state and the value of the *SparseCols* and *SparseRows* properties.

See also *GetActiveDim* method, *GetMemberAsString* method, *GetMemberAsVariant* method, *SparseCols* property, *SparseRows* property

GetDimensionName

GetDimensionName returns the display name for a dimension.

```
function GetDimensionName(iDim: Integer): String;
```

Description Call *GetDimensionName* to obtain a name to display as a label for the dimension. The *iDim* parameter is the absolute index of the dimension, where 0 indicates the first active dimension in the Decision cube, 1 indicates the second active dimension, and so on. To convert from a row-specific or column-specific dimension index into an absolute dimension index, use the *GetActiveDim* method.

The name of a dimension is often the name of the field the dimension represents. The dimension name can be changed in the Decision cube Editor or by setting the *Name* property of the dimension in the dimension map of the Decision cube.

See also *GetActiveDim* method, *TCustomDataStore.DimensionMap* property, *TDimensionItem.Name* property

GetGroup

GetGroup indicates where a dimension appears in the current pivot state.

```
type TDimGroup = (dgRow, dgCol, dgSum, dgPage, dgNone);

function GetGroup(iDim: Integer) : TDimGroup;
```

Description Call *GetGroup* to determine how the dimension specified by the *iDim* parameter is used in the current pivot state. The *iDim* parameter is the absolute index of the dimension, where 0 is the first active dimension, 1 is the second active dimension, and so on.

If the dimension is a row dimension (displayed to the left of a Decision grid or on the y-axis of a Decision graph), *GetGroup* returns *dgRow*. If it is a column dimension (displayed at the top of a Decision grid or on the x-axis of a Decision graph), *GetGroup* returns *dgCol*. If the dimension is a paged dimension (not appearing in the Decision control but limiting the data to a single value along the dimension), *GetGroup* returns *dgPage*. *GetGroup* never returns *dgSum* or *dgNone*.

Note To determine the position of the dimension within the row dimensions or column dimensions, use the *GetIndex* method.

See also *GetActiveDim* method, *GetIndex* method

GetGroupExtent

GetGroupExtent returns the range of cell indexes for a single value of a dimension.

```
type
  TDimRange = Record
    First: Integer;
    Last: Integer;
  end;

  TDimGroup = (dgRow, dgCol, dgSum, dgPage, dgNone);

function GetGroupExtent(dimGroup: TDimGroup; Index: Integer; Cell: Integer): TDimRange;
```

Description Call *GetGroupExtent* to find the range of cells in a Decision grid that have the same value on a given dimension as the cell specified by the *Cell* parameter. *Cell* is the index of the row or column containing the cell, where 0 specifies the first row or column in the data area of a Decision grid. If *dimGroup* is *dgRow*, *Cell* identifies a row of the grid. If *dimGroup* is *dgCol*, *Cell* identifies a column of the grid. Do not use other values for *dimGroup*. *Index* specifies the row-specific or column-specific index of the dimension.

For example, if a Decision grid has two row dimensions listed in the first two columns of the Decision grid, the line

```
MyRange := GetGroupExtent(dgRow, 0, 3);
```

sets *MyRange* to the range of row indexes for cells that have the same value on the first row dimension as the third cell. The number of cells in this range is the same as the value returned by *GetDimensionMemberCount* for the second open row dimension. The line

```
MyRange := GetGroupExtent(dgRow, 1, 3);
```

sets *MyRange* to the range from 3 to 3, because the second row dimension is the innermost row dimension, so each value is represented by a single row.

Note If the dimension includes subtotals, the subtotal cell is included in the range returned by *GetGroupExtent*.

See also *GetDimensionMemberCount* method

GetIndex

GetIndex returns the row-specific or column-specific index of a dimension.

function GetIndex(iDim: Integer; bOpen: boolean): Integer;

Description Call *GetIndex* to obtain the row-specific index or column-specific index of a dimension from its index in the Decision cube. *GetIndex* returns 0 for the first row dimension or the first column dimension. *GetIndex* returns 1 for the second row dimension or the second column dimension, and so on. The *iDim* parameter is the absolute index of the dimension. If the *bOpen* parameter is *True*, *GetIndex* is the position of the dimension when only open dimensions are counted. If *bOpen* is *False*, all dimensions, including closed dimensions are counted.

Use the *GetGroup* method to determine whether the index returned by *GetIndex* is a row-specific index (*GetGroup* returns *dgRow*) or a column-specific index (*GetGroup* returns *dgCol*).

To obtain the index of a dimension in the Decision cube from its row-specific or column-specific index, use the *GetActiveDim* method.

See also *GetActiveDim* method, *GetGroup* method

GetMemberAsString

GetMemberAsString returns the label string for a dimension value.

function GetMemberAsString(iDim: Integer; ValueIndex: Integer): String;

Description Call *GetMemberAsString* to obtain the value indicated by the *ValueIndex* parameter along the dimension specified by the *iDim* parameter, formatted as a string.

The *iDim* parameter is the absolute index of the dimension, where 0 indicates the first active dimension in the Decision cube, 1 indicates the second active dimension, and so on. To convert from a row-specific or column-specific dimension index into an absolute dimension index, use the *GetActiveDim* method.

The *ValueIndex* parameter is the index of a value along the dimension, where 0 specifies the first value of the dimension and *GetDimensionMemberCount* returns the number of values along the dimension.

See also *GetActiveDim* method, *GetDimensionMemberCount* method, *GetDimensionName* method, *GetMemberAsVariant* method

GetMemberAsVariant

GetMemberAsVariant returns a value along a dimension.

```
function GetMemberAsVariant(iDim: Integer; ValueIndex: Integer): Variant;
```

Description Call *GetMemberAsVariant* to obtain the value indicated by the *ValueIndex* parameter along the dimension specified by the *iDim* parameter.

The *iDim* parameter is the absolute index of the dimension, where 0 indicates the first active dimension in the Decision cube, 1 indicates the second active dimension, and so on. To convert from a row-specific or column-specific dimension index into an absolute dimension index, use the *GetActiveDim* method.

The *ValueIndex* parameter is the index of a value along the dimension, where 0 specifies the first value of the dimension and *GetDimensionMemberCount* returns the number of values along the dimension.

See also *GetActiveDim* method, *GetDimensionMemberCount* method, *GetDimensionName* method, *GetMemberAsString* method

GetRowState

GetRowState indicates whether the adjacent dimensions are open or closed.

```
type
  TRowStates = (rcNextOpen, rcPrevOpen, rcNextClosed, rcPrevClosed);
  TRowState = set of TRowStates;

function GetRowState(iDim: Integer): TRowState;
```

Description Call *GetRowState* to determine the state of the dimensions that are adjacent to a dimension in the current pivot state. The *iDim* parameter specifies the absolute index of the dimension whose neighbors are to be checked. When *iDim* is 0, the dimension is the first active dimension in the Decision cube. When *iDim* is 1, it is the second active dimension, and so on.

The return value includes *rcPrevOpen* if the previous dimension is open or *rcPrevClosed* if the previous dimension is closed. The return value includes *rcNextOpen* if the following dimension is open or *rcNextClosed* if it is closed. The first row or column dimension does not include either *rcPrevOpen* or *rcPrevClosed*. The last row or column does not include either *rcNextOpen* or *rcNextClosed*.

Note Do not be confused by the name of the *GetRowState* method. It provides information about the adjacent column dimensions if *iDim* specifies a

column dimension, or information about the adjacent row dimensions if *iDim* specifies a row dimension.

See also *GetState* method

GetState

GetState returns the state of a dimension.

```
type TDimState = (dmClosed, dmOpen, dmDrilled, dmPaged, dmNone);
function GetState(iDim: Integer): TDimState;
```

Description Call *GetState* to determine whether a specified dimension is open, closed, or drilled. The *iDim* parameter is the absolute index of the dimension, where 0 specifies the first active dimension in the Decision cube, 1 specifies the second active dimension, and so on. *GetState* returns one of the following values:

Value	Meaning
dmClosed	The dimension is closed. None of its values appear in the Decision control.
dmOpen	The dimension is open. All of its values appear in the Decision control.
dmDrilled	The dimension is drilled. Only a single value (or a summary over all values) along the dimension appears in the Decision control.
dmPaged	The dimension can only display a single value or summarize all values together.
dmNone	The dimension is not active.

Note The state *dmDrilled* appears the same as *dmPaged* in a Decision control. However, the state of drilled dimensions can be changed to open or closed, while paged dimensions can't change their state. Paged dimensions use considerably less memory than other types.

See also *GetRowState* method, *GetValue* method, *ToggleDimIndex* method

GetSummaryName

GetSummaryName returns the name of an active summary in the Decision cube.

```
function GetSummaryName(iSum: Integer): String;
```

Description Call *GetSummaryName* to determine the name of a summary. The *iSum* parameter is the index of the summary from the list of active summaries available from the Decision cube. Use *GetSummaryName* to determine the name of the possible summaries that can be set using the *SetCurrentSummary* method.

See also *SetCurrentSummary* method, *TCustomDataStore.SummaryCount* property

GetValue

GetValue returns the drill value of a drilled or paged dimension.

```
function GetValue(iDim: Integer): Integer;
```

Description Call *GetValue* to determine the value focused on by a drilled or paged dimension. Drilled dimensions and paged dimensions display only the data for a single field value along the dimension. *GetValue* returns the index of that single value. The value itself can be obtained from this index by calling the *GetMemberAsString* or *GetMemberAsVariant* method.

The *iDim* parameter is the absolute index of the drilled or paged dimension, where 0 specifies the first active dimension in the Decision cube, 1 specifies the second active dimension, and so on.

Note If the dimension is drilled so that it displays only summaries over all the values in the dimension, *GetValue* returns –1. If the dimension specified by the *iDim* parameter is not drilled or paged, *GetValue* returns –1.

See also *DrillDimIndex* method, *GetMemberAsString* method, *GetMemberAsVariant* method, *GetState* method

GetValueArray

GetValueArray returns the values of all dimension fields over which a data cell is summarized.

```
function GetValueArray(ACol, ARow: Integer; var ValueArray: TSmallIntArray): boolean;
```

Description Call *GetValueArray* to obtain the coordinates of a data cell from a Decision grid in the multidimensional array maintained by the Decision cube. The *ACol* and *ARow* parameters represent the indexes of the column and row containing the cell, where (0,0) specifies the upper left data cell, not the top left cell in the grid. The *ValueArray* parameter returns the indexes of the field values along each active dimension. An index of –1 indicates that the cell represents a subtotal across all values in the corresponding dimension. Use the *GetMemberAsString* or *GetMemberAsVariant* method to convert the indexes in the *ValueArray* parameter into field values. *GetValueArray* returns a boolean indicating whether the function was successful.

See also *GetMemberAsString* method, *GetMemberAsVariant* method, *GetValueIndex* method

GetValueIndex

GetValueIndex returns the index of a single dimension value that corresponds to a cell in a Decision grid.

```
type TDimGroup = (dgRow, dgCol, dgSum, dgPage, dgNone);

function GetValueIndex(dimGroup: TDimGroup; Index: Integer; Cell: Integer;
  var isBreak: boolean; var isSum: boolean) : Integer;
```

Description Call *GetValueIndex* to obtain the coordinate of a cell in a Decision grid along a the dimension specified by the *Index* parameter. The *dimGroup* parameter indicates whether the dimension is a row dimension (*dgRow*) or a column dimension (*dgCol*). Do not call *GetValueIndex* with *dimGroup* set to a dimension group other than *dgRow* or *dgCol*. The *Index* parameter is the row-specific or column-specific index of the dimension. The first row or column dimension has an *Index* of 0, the second dimension has an *Index* of 1, and so on. The *Cell* parameter is the index of the cell row or column, where 0 specifies the first row or column in the data region of the grid.

In addition to returning the index of the dimension value, *GetValueIndex* provides additional information about the position of *Cell* within the cells for the dimension. *GetValueIndex* sets *isBreak* to *True* when *Cell* is the first cell in the dimension group with the returned value on the specified dimension. *GetValueIndex* sets *iSum* to *True* when *Cell* represents a subtotal over the dimension.

See also *GetActiveDim* method, *GetGroupExtent* method, *GetValueArray* method

MoveDimIndexes

MoveDimIndexes changes the position of a dimension.

```
type TDimGroup = (dgRow, dgCol, dgSum, dgPage, dgNone);

procedure MoveDimIndexes(SDimGroup, DDimGroup: TDimGroup; SIndex, DIndex: Integer;
  bOpen:boolean);
```

Description Call *MoveDimIndexes* to change the pivot state of the Decision grid. *MoveDimIndexes* moves the dimension indicated by the *SIndex* and *SDimGroup* parameters to the position indicated by the *DIndex* and *DDimGroup* parameters.

The *SDimGroup* and *DDimGroup* parameters specify whether the dimension position is along the rows of a Decision grid (*SDimGroup* or *DDimGroup* is *dgRow*), or across the columns of the Decision grid (*SDimGroup* or *DDimGroup* is *dgCol*). The *SIndex* and *DIndex* parameters indicate the index of the dimension position among the row dimensions or column dimensions, where 0 specifies the first position, 1 specifies the second position, and so on. The *bOpen* parameter specifies whether closed dimensions are counted when interpreting the *SIndex* and *DIndex* parameters. When *bOpen* is *True*, only open dimensions are counted.

See also *GetActiveDim* method, *SwapDimIndexes* method

OpenDimIndexLeft

OpenDimIndexLeft opens the first closed dimension that appears before a specified dimension.

```
type TDimGroup = (dgRow, dgCol, dgSum, dgPage, dgNone);

procedure OpenDimIndexLeft(dimGroup: TDimGroup; Index: Integer; bOpen: boolean);
```

Description Call *OpenDimIndexLeft* to reopen a closed dimension so that its values appear in the Decision control.

OpenDimIndexLeft opens the first closed dimension before the dimension identified by the *Index* parameter. If the *dimGroup* parameter is *dgRow*, *Index* identifies a row dimension, where 0 specifies the first row dimension, 1 specifies the second row dimension, and so on. If the *dimGroup* parameter is *dgCol*, *Index* identifies a column dimension, starting with an index of 0 for the first column dimension. No other values for *dimGrouop* should be used with the *OpenDimIndexLeft* method. To open the last closed row or column dimension, set *Index* to –1. The *bOpen* parameter specifies whether closed dimensions are counted when interpreting *Index*. Only open dimensions are counted when *bOpen* is *True*.

See also *CloseDimIndexRight* method, *GetActiveDim* method, *GetRowState* method, *OpenDimIndexRight* method, *ToggleDimIndex* method

OpenDimIndexRight

OpenDimIndexRight opens the first closed dimension that appears after a specified dimension.

```
type TDimGroup = (dgRow, dgCol, dgSum, dgPage, dgNone);

procedure OpenDimIndexRight(dimGroup: TDimGroup; Index: Integer; bOpen: boolean);
```

Description Call *OpenDimIndexRight* to reopen a closed dimension so that its values appear in the Decision control.

OpenDimIndexRight opens the first closed dimension after the dimension identified by the *Index* parameter. If the *dimGroup* parameter is *dgRow*, *Index* identifies a row dimension, where 0 specifies the first row dimension, 1 specifies the second row dimension, and so on. If the *dimGroup* parameter is *dgCol*, *Index* identifies a column dimension, starting with an index of 0 for the first column dimension. No other values for *dimGrouop* should be used with the *OpenDimIndexRight* method. To open the first closed row or column dimension, set *Index* to –1. The *bOpen* parameter specifies whether closed dimensions are counted when interpreting *Index*. Only open dimensions are counted when *bOpen* is *True*.

See also *CloseDimIndexRight* method, *GetActiveDim* method, *GetRowState* method, *OpenDimIndexLeft* method, *ToggleDimIndex* method

SetCurrentSummary

SetCurrentSummary specifies which summary is provided by the Decision source.

```
procedure SetCurrentSummary(Value: Integer);
```

Description The Decision cube maintains a set of active summaries in its cache. The Decision source can only present one of those summaries to Decision controls. Call *SetCurrentSummary* to specify which of the active summaries is

surfaced by the Decision source. The *Value* parameter specifies the index of the summary in the list of active summaries maintained by the Decision cube.

To obtain the names of the active summaries from which the current summary is chosen, use the *GetSummaryName* method.

See also *GetSummaryName* method, *TCustomDataStore.CurrentSummary* property

SwapDimIndexes

SwapDimIndexes interchanges the positions of two dimensions.

```
type TDimGroup = (dgRow, dgCol, dgSum, dgPage, dgNone);

procedure SwapDimIndexes(SDimGroup, DDimGroup: TDimGroup; SIndex, DIndex: Integer;
  bOpen:boolean);
```

Description Call *SwapDimIndexes* to change the pivot state of the Decision grid. *SwapDimIndexes* interchanges the positions of the dimension indicated by the *SIndex* and *SDimGroup* parameters and the dimension indicated by the *DIndex* and *DDimGroup* parameters.

The *SDimGroup* and *DDimGroup* parameters specify whether the dimension position is along the rows of a Decision grid (*SDimGroup* or *DDimGroup* is *dgRow*), or across the columns of the Decision grid (*SDimGroup* or *DDimGroup* is *False*). The *SIndex* and *DIndex* parameters indicate the index of the dimension position among the row dimensions or column dimensions, where 0 specifies the first position, 1 specifies the second position, and so on. The *bOpen* parameter specifies whether closed dimensions are considered when interpreting the *SIndex* and *DIndex* parameters. When *bOpen* is *True*, only open dimensions are counted.

See also *GetActiveDim* method, *MoveDimIndexes* method

ToggleDimIndex

ToggleDimIndex changes whether a dimension is open, closed, or drilled.

```
type TDimGroup = (dgRow, dgCol, dgSum, dgPage, dgNone);

procedure ToggleDimIndex(dimGroup: TDimGroup; Index: Integer; bOpen: boolean);
```

Description Call *ToggleDimIndex* to open a closed dimension, close an open dimension, or expand a drilled dimension.

When *dimGroup* is *dgRow*, the *Index* parameter specifies the row-specific index of the dimension. When *dimGroup* is *dgCol*, *Index* specifies the column-specific index of the dimension. The first row or column dimension has an *Index* of 0, the second dimension has an *Index* of 1, and so on. The *bOpen* parameter specifies whether closed dimensions should be counted when interpreting the *Index* parameter. When *bOpen* is *True*, only open dimensions are counted.

See also *CloseDimIndexRight* method, *DrillDimIndex* method, *GetActiveDim* method, *GetState* method, *OpenDimIndexLeft* method, *OpenDimIndexRight* method

TDirectoryListBox

Unit: FileCtrl | Palette: Win3.1

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomListBox • TDirectoryListBox

TDirectoryListBox is a list box control that is aware of the directory structure of the current drive.

Description *TDirectoryListBox* implements the generic behavior introduced in *TCustomListbox*. *TDirectoryListBox* publishes many of the properties inherited from *TCustomListBox*, and introduces any new behavior to allow the user to change directories or drives.

For other specialized list boxes see other descendant classes of *TCustomListBox*, or derive from *TCustomListBox*.

See also *TDriveComboBox* object, *TFileListBox* object, *TFilterComboBox* object

Properties

The following table lists the properties of *TDirectoryListBox*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Canvas	public, read-only		TCustomListBox
CaseSensitive	public, read-only		TDirectoryListBox
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
Columns	published	0	TCustomListBox
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl

Property	Visibility	Default	Documented in
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
Directory	public		TDirectoryListBox
DirLabel	published		TDirectoryListBox
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Drive	public		TDirectoryListBox
Enabled	published	True	TControl
FileList	published		TDirectoryListBox
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
ImeMode	published	imDontCare	TWinControl
ImeName	published		TWinControl
IntegralHeight	published	False	TCustomListBox
ItemHeight	published		TCustomListBox
ItemIndex	public		TCustomListBox
Items	public		TCustomListBox
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
PreserveCase	public, read-only		TDirectoryListBox
SelCount	public, read-only		TCustomListBox
Selected	public		TCustomListBox
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent

Property	Visibility	Default	Documented in
Top	published		TControl
TopIndex	public		TCustomListBox
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

CaseSensitive

CaseSensitive determines if the disk system supports case-sensitive directory information.

property CaseSensitive: Boolean;

Description Use *CaseSensitive* to find whether or not the current file system supports case-sensitive file names. *CaseSensitive* returns *True* if it does and *False* if it does not.

See also *DisplayCase* method, *FileCompareText* method, *PreserveCase* property

Directory

Directory determines the current directory for the directory list box control.

property Directory: **string**;

Description The directory list box displays the value of the *Directory* property as the current directory in the list box. Examine the example to see how a directory list box and a file list box can work together through their *Directory* properties.

Example For a file list box and a directory list box on a form, this code changes the current directory in the directory list box and displays the files in that directory in the file list box when the user changes directories using the directory list box:

```
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

Note If the form contains a file list box, set the *FileList* property to have the file list box automatically display the files from the directory specified by the *Directory* property.

See also *DirLabel* property, *Drive* property, *FileList* property

DirLabel

DirLabel provides a simple way to display the current directory as the caption of a label control.

property DirLabel: TLabel;

Description Use *DirLabel* to specify the label to update with the current directory. When the current directory changes in the directory list box, the change is reflected in the caption of the label.

See also *Caption* property, *Directory* property, *DirList* property, *FileEdit* property, *FileList* property

Drive

Drive determines the drive for which the list box displays the directory structure.

property Drive: Char;

Description When the value of *Drive* changes, the *Directory* value changes also to the current directory on the specified drive.

Example The following example assumes that a drive combo box, a file list box, and a directory list box are on a form. This code changes the drive displayed in the drive combo box, displays the current directory of the selected drive in the directory list box, and displays the files in the current directory of the selected drive in the file list box when the user selects a drive in the drive combo box:

```
procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See also *Directory* property, *DirList* property, *Text* property

FileList

FileList provides a simple way to connect a directory list box with a file list box.

property FileList: *TFileListBox*;

Description Use *FileList* to specify the file list box in which to display the files of the current directory.

Set *FileList* to the file list box that will display the files in the directory. Set the *Directory* property of the directory list box.

Example This example uses a button, an edit box, a label, a drive combo box, a directory list box, a file list box, and a filter combo box on a form. When the

user clicks the button, the rest of the controls on the form begin working together like the controls in an open or save dialog box.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  DriveComboBox1.DirList := DirectoryListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
end;
```

See also *DirLabel* property, *Directory* property, *TDriveComboBox.DirList* property, *TFileListBox.FileEdit* property, *TFilterComboxBox.Filter* property, *TFileListBox.Mask* property, *TFilterComboBox.Mask* property

PreserveCase

PreserveCase indicates whether the file system preserves the case of file names when writing to disk.

```
property PreserveCase: Boolean;
```

Description Use *PreserveCase* to find whether or not the current file system preserves the case-sensitivity of file names. *PreserveCase* returns *True* if it does and *False* if it does not.

See also *CaseSensitive* property, *DisplayCase* method

Events

The following table lists the events of *TDirectoryListBox*:

Event	Visibility	Documented in
OnChange	published	TDirectoryListBox
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl

Event	Visibility	Documented in
OnMouseUp	published	TControl
OnStartDrag	published	TControl

D

OnChange

OnChange occurs when the user selects a new directory with the mouse or keyboard when the user moves the selection bar and presses *Enter*.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnChange: TNotifyEvent;
```

Description Use *OnChange* to write an event handler to process an action when a new directory is selected.

Methods

The following table lists the methods of *TDirectoryListBox:*

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TCustomListBox
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TDirectoryListBox
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TDirectoryListBox
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject

Method	Visibility	Documented in
DisplayCase	public	TDirectoryListBox
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FileCompareText	public	TDirectoryListBox
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetItemPath	public	TDirectoryListBox
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
ItemAtPos	public	TCustomListBox
ItemRect	public	TCustomListBox
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
OpenCurrent	public	TDirectoryListBox
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl

Method	Visibility	Documented in
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TDirectoryListBox
UpdateControlState	public	TWinControl

Create

Create instantiates a directory list box.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to instantiate a directory list box at runtime. For directory list boxes created at design time, *Create* is called automatically.

Create allocates memory for a directory list box, and calls the inherited *Create*. Then it set the following properties:

- *Width* to 145.
- *Style* to *lbOwnerDrawFixed*.
- *Sorted* to *False*.

Then it creates and loads bitmaps for the Open, Close and Current folders and initializes the directory to the current directory on the default drive.

See also *TCustomListBox.Create* method, *TCustomListBox.Sorted* method, *TCustomListBox.Style* method

Destroy

Destroy disposes of an instance of a directory list box.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, an application should call *Free*. Free verifies that the list box object has not already been freed before it calls *Destroy*.

Destroy frees the bitmaps used for the Open, Close and Current folders, and then calls the inherited *Destroy*.

See also *TCustomListBox.Destroy* method

DisplayCase

DisplayCase either preserves or converts the case of the string passed to it.

```
function DisplayCase(const S: String): String;
```

Description *DisplayCase* converts *S* to all lower case, using the currently installed language driver, if neither *PreserveCase* nor *CaseSensitive* are True. Otherwise it simply returns *S* unmodified.

See also *CaseSensitive* property, *PreserveCase* property

FileCompareText

FileCompareText compares two strings taking into account the case sensitivity of the file system.

```
function FileCompareText(const A, B: String): Integer;
```

Description *FileCompareText* compares the strings passed to it as *A* and *B*, if *CaseSensitive* is *False*. Otherwise it uses *AnsiCompareStr* to compare the strings.

See also *AnsiCompareStr* function, *CaseSensitive* property

GetItemPath

GetItemPath returns the path of a directory in the directory list box.

```
function GetItemPath(Index : Integer): string;
```

Description Use *GetItemPath* to get the path of a directory by specifying the directory with the *Index* value. The first directory in the list has an index value of 0.

Example This example uses a directory list box, a button, and a label on a form. When the user selects a directory in the directory list box and clicks the button, the selected directory opens, and the path of the second directory displayed in the list box appears as the caption of the label.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  DirectoryListBox1.OpenCurrent;
  Label1.Caption := DirectoryListBox1.GetItemPath(1);
end;
```

See also *Directory* property, *Drive* property, *OpenCurrent* method

OpenCurrent

OpenCurrent opens the selected directory.

procedure OpenCurrent;

Description *OpenCurrent* opens the directory selected in the directory list box, as if the user had double-clicked the directory.

Example This example uses a directory list box, a button, and a label on a form. When the user selects a directory in the directory list box and clicks the button, the selected directory opens, and the path of the second directory displayed in the list box appears as the caption of the label.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  DirectoryListBox1.OpenCurrent;
  Label1.Caption := DirectoryListBox1.GetItemPath(1);
end;
```

Update

Update refreshes the directory list.

procedure Update;

Description *Update* updates and refreshes the directory list for the directory list box control to reflect the current settings.

Example The following sample code sets the directory of *DirectoryListBox1* to C:\TEMP when the form is created. When *Button1* is pressed, a subdirectory called MYDIR is added to C:\TEMP, but note that it is not updated in *DirectoryListBox1* until *Button2* is pressed and *Update* is called.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  MkDir('c:\temp\mydir');
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  DirectoryListBox1.Directory := 'c:\temp';
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  DirectoryListBox1.Update;
end;
```

TDrawGrid

Unit: Grids | Palette: Additional

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomControl • TCustomGrid • TDrawGrid

TDrawGrid is a grid control that permits the display of information in column and row format.

Description

Add a *TDrawGrid* object to a form to present arbitrary information in a tabular format. *TDrawGrid* provides many properties to control the appearance of the grid, as well as events and methods that take advantage of the tabular organization of the grid in responding to user actions.

TDrawGrid introduces several events and methods to allow applications to customize the appearance and behavior of the grid.

If the cells of the grid contain text strings that represent objects, consider using a *TStringGrid* object instead. If the grid displays field values from the records in a dataset, use *TDBGrid* instead.

See also *TDBGrid* object, *TStringGrid* object

Properties

The following table lists the properties of *TDrawGrid*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
BorderStyle	published	bsSingle	TCustomGrid
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Canvas	public, read-only		TCustomControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Col	public		TCustomGrid
ColCount	published	5	TCustomGrid
Color	published	clWindow	TControl
ColWidths	public		TCustomGrid
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent

Property	Visibility	Default	Documented in
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DefaultColWidth	published	64	TCustomGrid
DefaultDrawing	published	True	TCustomGrid
DefaultRowHeight	published	24	TCustomGrid
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
EditorMode	public		TCustomGrid
Enabled	published	True	TControl
FixedColor	published	clBtnFace	TCustomGrid
FixedCols	published	1	TCustomGrid
FixedRows	published	1	TCustomGrid
Font	published		TControl
GridHeight	public, read-only		TCustomGrid
GridLineWidth	published	1	TCustomGrid
GridWidth	public, read-only		TCustomGrid
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
LeftCol	public		TCustomGrid
Name	published		TControl
Options	published	[goFixedVertLine, goFixedHorzLine, goVertLine, goHorzLine, goRangeSelect]	TCustomGrid
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
Row	public		TCustomGrid
RowCount	published	5	TCustomGrid

Property	Visibility	Default	Documented in
RowHeights	public		TCustomGrid
ScrollBars	published	ssBoth	TCustomGrid
Selection	public		TCustomGrid
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
TabStops	public		TCustomGrid
Tag	published	0	TComponent
Top	published		TControl
TopRow	public		TCustomGrid
VCLComObject	public		TComponent
Visible	published	True	TControl
VisibleColCount	published, read-only		TCustomGrid
VisibleRowCount	published, read-only		TCustomGrid
Width	published		TControl
WindowProc	public		TControl

Events

The following table lists the events of *TDrawGrid*:

Event	Visibility	Documented in
OnClick	published	TControl
OnColumnMoved	published	TDrawGrid
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnDrawCell	published	TDrawGrid
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnGetEditMask	published	TDrawGrid
OnGetEditText	published	TDrawGrid
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnRowMoved	published	TDrawGrid

Event	Visibility	Documented in
OnSelectCell	published	TDrawGrid
OnSetEditText	published	TDrawGrid
OnStartDrag	published	TControl
OnTopLeftChanged	published	TDrawGrid

OnColumnMoved

OnColumnMoved occurs immediately after the position of a column changes.

```
type TMovedEvent = procedure (Sender: TObject; FromIndex, ToIndex: Longint) of object;

property OnColumnMoved: TMovedEvent;
```

Description Use *OnColumnMoved* to perform special processing when the position of a grid column changes. Columns can be moved programmatically or by user manipulation. The *FromIndex* parameter is the old index of the column, and the *ToIndex* parameter is the new index. *OnColumnMoved* does not occur unless the *Options* property includes *goColMoving*.

See also *OnRowMoved* event, *Options* property

OnDrawCell

OnDrawCell occurs when a cell in the grid needs to be drawn.

```
type
  TGridDrawState = set of (gdSelected, gdFocused, gdFixed);
  TDrawCellEvent = procedure (Sender: TObject; Col, Row: Longint; Rect: TRect;
    State: TGridDrawState) of object;

property OnDrawCell: TDrawCellEvent;
```

Description Write an *OnDrawCell* event handler to draw the contents all the cells in the grid. Draw on the cell using the methods of the *Canvas* property. The *Rect* parameter indicates the location of the cell on the canvas. The *Col* and *Row* parameters indicate the column and row indexes of the cell that should be drawn. The *State* parameter indicates whether the cell has input focus, whether the cell is selected, and whether the cell is a fixed (nonscrolling) cell.

If the *OnDrawCell* event handler is not assigned, all cells in the draw grid will appear empty. If the *DefaultDrawing* property is *True*, the draw grid paints the background color of the cell before the *OnDrawCell* event, and draws a focus rectangle around the selected cell after the *OnDrawCell* event handler finishes drawing the contents of the cell. If the *DefaultDrawing* property is *False*, the *OnDrawCell* event handler should paint the background of the cell and provide all visual indication of selection and focus.

See also *Canvas* property, *DefaultDrawing* property

OnGetEditMask

OnGetEditMask occurs when the in-place editor requests an edit mask.

```
type TGetEditEvent = procedure (Sender: TObject; ACol, ARow: Longint; var Value: string)
  of object;

property OnGetEditMask: TGetEditEvent;
```

Description Write an *OnGetEditMask* event handler to provide the in-place editor with an edit mask that describes the valid input strings a user can type into a cell. Set the *Value* parameter to the string that describes the edit mask for the cell specified by the *ACol* and *ARow* parameters. If the cell does not have an edit mask, set the *Value* parameter to an empty string.

OnGetEditMask does not occur unless the *Options* property includes *goEditing*.

See also *Options* property, *TCustomMaskEdit.EditMask* property

OnGetEditText

OnGetEditText occurs when the in-place editor requests the value of a cell.

```
type TGetEditEvent = procedure (Sender: TObject; ACol, ARow: Longint; var Value: string)
  of object;

property OnGetEditText: TGetEditEvent;
```

Description Write an *OnGetEditText* event handler to provide the in-place editor with a string representing the contents of the cell. Set the *Value* parameter to the string for the cell specified by the *ACol* and *ARow* parameters. When the cell enters edit mode, the contents of the cell are drawn as the *Value* string returned by the *OnGetEditText* event handler. This image need not match the appearance of the cell when it is not in edit mode, which is drawn using the *OnDrawCell* event handler.

OnGetEditText does not occur unless the *Options* property includes *goEditing*.

See also *OnDrawCell* event, *OnSetEditText* event

OnRowMoved

OnRowMoved occurs immediately after the position of a row changes.

```
type TMovedEvent = procedure (Sender: TObject; FromIndex, ToIndex: Longint) of object;

property OnRowMoved: TMovedEvent;
```

Description Write an *OnRowMoved* event handler to provide special processing whenever a row in the grid changes position. Rows can be moved programmatically or by user manipulation. The *FromIndex* parameter is the old index of the row, and the *ToIndex* parameter is the new index.

OnRowMoved does not occur unless the *Options* property includes *goRowMoving*.

See also *OnColumnMoved* event, *Options* property

OnSelectCell

OnSelectCell occurs before a cell in the grid is selected.

```
type TSelectCellEvent = procedure (Sender: TObject; Col, Row: Longint;
  var CanSelect: Boolean) of object;

property OnSelectCell: TSelectCellEvent;
```

Description Write an *OnSelectCell* event handler to specify whether any particular cell in the grid can be selected. The *Col* and *Row* parameters indicate the column and row indexes of the cell that is about to be selected. Set the *CanSelect* parameter to *False* to prevent the cell being selected.

See also *Col* property, *Row* property, *TabStops* property

OnSetEditText

OnSetEditText occurs when the user has finished editing the value of a cell.

```
type TSetEditEvent = procedure (Sender: TObject; ACol, ARow: Longint; const Value: string)
  of object;

property OnSetEditText: TSetEditEvent;
```

Description Write an *OnSetEditText* event handler to perform any special processing of the text edited by the user in an in-place editor. For example, use the *OnSetEditText* event to retrieve and store the value of a cell so that it can be displayed in an *OnDrawCell* event handler. The *Value* parameter is the new value for the cell specified by the *ACol* and *ARow* parameters.

OnSetEditText does not occur unless the *Options* property includes *goEditing*.

See also *OnDrawCell* event, *OnGetEditText* event, *Options* property

OnTopLeftChanged

OnTopLeftChanged occurs immediately after the *TopRow* property or the *LeftCol* property changes.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnTopLeftChanged: TNotifyEvent;
```

Description Use *OnTopLeftChanged* to perform special processing when the non-fixed cells in the grid are scrolled.

See also *LeftCol* property, *TopRow* property

Methods

The following table lists the methods of *TDrawGrid*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
CellRect	public	TDrawGrid
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ColumnMoved	protected	TDrawGrid
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomGrid
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TCustomGrid
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
DrawCell	protected	TDrawGrid
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetEditMask	protected	TDrawGrid
GetEditText	protected	TDrawGrid

Method	Visibility	Documented in
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
MouseCoord	public	TCustomGrid
MouseToCell	public	TDrawGrid
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
RowMoved	protected	TDrawGrid
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SelectCell	protected	TDrawGrid
SendToBack	public	TControl
SetBounds	public	TWinControl
SetEditText	protected	TDrawGrid
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl

Method	Visibility	Documented in
TopLeftChanged	protected	TDrawGrid
Update	public	TWinControl
UpdateControlState	public	TWinControl

CellRect

CellRect returns the screen coordinates of a cell in the grid.

```
function CellRect(ACol, ARow: Longint): TRect;
```

Description Call *CellRect* to obtain the boundaries, in pixels, of the cell in the column and row indicated by the *ACol* and *ARow* parameters. The top left cell is specified when *ACol* and *ARow* are set to zero.

If the indicated cell is not visible, *CellRect* returns an empty rectangle.

See also *MouseToCell* method, *TRect* type

ColumnMoved

ColumnMoved generates an *OnColumnMoved* event.

```
procedure ColumnMoved(FromIndex, ToIndex: Longint); override;
```

Description *ColumnMoved* is called immediately after a column in the grid has moved. It overrides the inherited method to generate an *OnColumnMoved* event. Descendants of *TDrawGrid* can override this method to make internal adjustments or to block the *OnColumnMoved* event.

See also *OnColumnMoved* event

DrawCell

DrawCell generates an *OnDrawCell* event.

```
type TGridDrawState = set of (gdSelected, gdFocused, gdFixed);
```

```
procedure DrawCell(ACol, ARow: Longint; ARect: TRect; AState: TGridDrawState); override;
```

Description *DrawCell* is called automatically whenever a cell in the grid is drawn. If the *DefaultDrawing* property is *True*, the background of the cell is painted before *DrawCell* is called, and the 3D effects of the fixed cells or the focus rectangle around the focused cell are drawn after *DrawCell*. Descendants of *TDrawGrid* can override this method to draw the contents of the cell or to block the *OnDrawCell* event.

See also *DefaultDrawing* property, *FixedCols* property, *FixedRows* property, *OnDrawCell* event

GetEditMask

GetEditMask generates an *OnGetEditMask* event.

```
function GetEditMask(ACol, ARow: Longint): string; override;
```

Description The inplace editor calls *GetEditMask* to obtain any input mask associated with the cell specified by the *ACol* and *ARow* parameters. *GetEditMask* returns the *Value* parameter set by the *OnGetEditMask* event handler. If there is no *OnGetEditMask* event handler, *GetEditMask* returns an empty string, indicating that there is no edit mask.

See also *OnGetEditMask* event

GetEditText

GetEditText generates an *OnGetEditText* event.

```
function GetEditText(ACol, ARow: Longint): string; override;
```

Description The inplace editor calls *GetEditText* to obtain the text for the indicated cell. *GetEditText* returns the *Value* parameter set by the *OnGetEditText* event handler. If there is no *OnGetEditText* event handler, *GetEditText* returns an empty string.

See also *OnGetEditText* event

MouseToCell

MouseToCell returns the column and row of the cell at the position with screen coordinates (X,Y).

```
procedure MouseToCell(X, Y: Integer; var ACol, ARow: Longint);
```

Description Call *MouseToCell* to convert from grid-relative screen coordinates to row and column indexes. The X and Y parameters are the screen coordinates of the point to be converted. *MouseToCell* returns the *ACol* parameter as the number of the column over the point (X,Y), and the *ARow* parameter as the number of the row.

Usually the *MouseToCell* method is used in a mouse event handler, which supplies the mouse coordinates as the X and Y parameters of the method call.

See also *CellRect* method

RowMoved

RowMoved generates an *OnRowMoved* event.

```
procedure RowMoved(FromIndex, ToIndex: Longint); override;
```

Description *RowMoved* is called immediately after a row in the grid changes position. It overrides the inherited method to generate an *OnRowMoved* event.

Descendants of *TDrawGrid* can override this method to make internal adjustments or to block the *OnRowMoved* event.

See also *OnRowMoved* event

SelectCell

SelectCell determines whether a particular cell in the grid can be selected.

```
function SelectCell(ACol, ARow: Longint): Boolean; override;
```

Description Grid controls call *SelectCell* internally before attempting to move selection to a the cell identified by the *ACol* and *ARow* parameters. *ACol* and *ARow* are the column and row indexes for the cell, where the first column has index 0, and the top row has index 0.

SelectCell generates an *OnSelectCell* event. This method then returns *True*, unless the *OnSelectCell* event handler indicates that the cell should not be selected. Descendants of *TDrawGrid* can override this method to check the properties of the information displayed in the cell or block the *OnSelectCell* event.

See also *OnSelectCell* event

SetEditText

SetEditText generates an *OnSetEditText* event.

```
procedure SetEditText(ACol, ARow: Longint; const Value: string); dynamic;
```

Description *SetEditText* is called by the inplace editor so that the grid can store the new value for the current cell. *SetEditText* overrides the inherited method to generate an *OnSetEditText* event. Descendants of *TDrawGrid* can override this method to update an internal representation of the cell's value, or to block the *OnSetEditText* event.

See also *OnSetEditText* event

TopLeftChanged

TopLeftChanged generates an *OnTopLeftChanged* event.

```
procedure TopLeftChanged; dynamic;
```

Description Grid controls call *TopLeftChanged* internally, whenever the *LeftCol* or *TopRow* property changes. After calling the inherited method to update the position of the inplace editor, *TopLeftChanged* generates an *OnTopLeftChanged* event.

See also *LeftCol* property, *OnTopLeftChanged* event, *TopRow* property

TDriveComboBox

Unit: FileCtrl | Palette: Win3.1

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomComboBox • TDriveComboBox

TDriveComboBox is a specialized combo box that displays all the drives available when the application runs.

Description

Add *TDriveComboBox* to a form to allow users to select a drive. Use *TDriveComboBox* along with *TFileListBox*, *TFilterComboBox*, and *TDirectoryListBox* to add full file selection capabilities to a form.

To add a standard Windows 95 file open or save dialog to an application, use *TOpenDialog* or *TSaveDialog* instead.

See also

TDirectoryListBox object, *TFileListBox* object, *TFilterComboBox* object, *TOpenDialog* object, *TSaveDialog* object

Properties

The following table lists the properties of *TDriveComboBox*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Canvas	public, read-only		TCustomComboBox
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent

Property	Visibility	Default	Documented in
DirList	published		TDriveComboBox
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Drive	public		TDriveComboBox
DroppedDown	public		TCustomComboBox
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
ImeMode	published	imDontCare	TWinControl
ImeName	published		TWinControl
ItemIndex	public		TCustomComboBox
Items	public		TCustomComboBox
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
SelLength	public		TCustomComboBox
SelStart	public		TCustomComboBox
SelText	public		TCustomComboBox
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent
Text	public		TControl
TextCase	published	tcLowerCase	TDriveComboBox
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

DirList

DirList connects the drive combo box to a directory list box.

```
property DirList: TDirectoryListBox;
```

Description Set *DirList* to a directory list box control that should update when the user changes the drive. When a new drive is selected in the drive combo box, the specified directory list box updates to display the directory structure and the current directory on the new drive.

Note If the form contains a file list box as well, it can be updated automatically using the *FileList* property of the directory list box object.

See also *TDirectoryListBox* object, *TFileListBox* object

Drive

Drive is the currently selected drive displayed in the edit portion of the combo box.

```
property Drive: Char;
```

Description Set *Drive* to initialize the combo box to a particular drive. Read *Drive* to obtain the drive that was selected by the user.

When the user uses the drive combo box to select a new drive, the selected drive becomes the value of the *Drive* property. The value of the *Text* property also changes to the new volume name when the *Drive* property value changes.

See also *Text* property

TextCase

TextCase determines if the volume name in the *Text* property appears in uppercase or lowercase.

```
type TTextCase = (tcLowerCase, tcUpperCase);

property TextCase: TTextCase;
```

Description Set *TextCase* to the case that should be used to represent the drive letter. These are the possible values:

Value	Meaning
tcLowerCase	The volume name specified in the *Text* property is displayed in lowercase letters.
tcUpperCase	The volume name specified in the *Text* property is displayed in uppercase letters.

Note Changes made to the *TextCase* property at design time don't appear until the application runs.

See also *Drive* property, *Text* property

Events

The following table lists the events of *TDriveComboBox*:

Event	Visibility	Documented in
OnChange	published	TCustomComboBox
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnDropDown	published	TCustomComboBox
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TDriveComboBox*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TCustomComboBox
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TDriveComboBox
CreateParented	public	TWinControl

Method	Visibility	Documented in
DefaultHandler	public	TObject
Destroy	public	TDriveComboBox
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl

Method	Visibility	Documented in
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SelectAll	public	TCustomComboBox
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

Create

Create creates and initializes an instance of *TDriveComboBox*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to create a *TDriveComboBox* object at runtime. Drive combo boxes placed on a form at design time are created automatically.

After allocating the memory for the combo box, *Create* fills the combo box with all the drives on the system, and initializes the *Drive* property to the current drive.

See also *Drive* property, *TCustomComboBox.Create* method

Destroy

Destroy destroys an instance of *TDriveComboBox*.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the drive combo box is not already freed, and only then calls *Destroy*.

See also *Free* method

TDSTableProducer

Unit: DBWeb

TObject • TPersistent • TComponent • TCustomContentProducer • TDSTableProducer

TDSTableProducer is the abstract base class for all objects that produce a set of HTML commands to display the records from a dataset in a tabular format.

Description Use *TDSTableProducer* as a base class when defining a class that produces an HTML table that represents the records of a dataset. *TDSTableProducer* introduces many properties to describe the appearance and data binding of the HTML table. The events and methods of *TDSTableProducer* allow an application to retrieve the database information and produce the HTML commands that describe the table.

See also *TDataSetTableProducer* object, *TQueryTableProducer* object

Properties

The following table lists the properties of *TDSTableProducer*:

Property	Visibility	Default	Documented in
Caption	public		TDSTableProducer
CaptionAlignment	public	caDefault	TDSTableProducer
Columns	public		TDSTableProducer
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DataLink	protected, read-only		TDSTableProducer
DataSet	public		TDSTableProducer
DataSource	protected		TDSTableProducer
DesignInfo	public		TComponent
Dispatcher	published		TCustomContentProducer
Editor	public		TDSTableProducer
Footer	public		TDSTableProducer
Header	public		TDSTableProducer
InternalDataSource	protected, read-only		TDSTableProducer
MaxRows	public	20	TDSTableProducer
Name	published		TComponent
Owner	public, read-only		TComponent
RowAttributes	public		TDSTableProducer

Property	Visibility	Default	Documented in
TableAttributes	public		TDSTableProducer
Tag	published	0	TComponent
VCLComObject	public		TComponent

Caption

Caption specifies the caption string that appears with the HTML table.

```
property Caption: string;
```

Description Set *Caption* to add a caption to the HTML table that represents the dataset. *Caption* indicates the text that should appear inside the HTML CAPTION tag.

See also *CaptionAlignment* property, *OnGetTableCaption* event

CaptionAlignment

CaptionAlignment specifies where the *Caption* string should appear relative to the HTML table.

```
type THTMLCaptionAlignment = (caDefault, caTop, caBottom);
```

```
property CaptionAlignment: THTMLCaptionAlignment;
```

Description Set *CaptionAlignment* to specify whether the *Caption* should appear above or below the HTML table. *CaptionAlignment* can be one of the following values:

Value	Meaning
caDefault	The Web browser can position the caption where it wants.
caTop	The caption should appear above the HTML table.
caBottom	The caption should appear below the HTML table.

CaptionAlignment is the value of the ALIGN option on the HTML CAPTION tag.

See also *Caption* property

Columns

Columns is an indexed collection of *THTMLTableColumn* objects that describe the bindings and display attributes of the columns in the HTML table.

```
property Columns: THTMLTableColumns;
```

Description Use *Columns* to read or set the field bindings and display attributes of the columns in the HTML table. Each *THTMLTableColumn* object specifies the display attributes of the corresponding column. Each column object also specifies the name of the field the column represents. The field binding of a

column designates a field within the dataset specified by the *DataSet* property.

Columns can be set at design time through a property editor, or programmatically at runtime.

See also *DataSet* property, *DefineFieldMap* method, *Editor* property, *RowAttributes* property, *SetColumns* method, *StoreColumns* method, *TableAttributes* property, *THTMLTableColumn* object, *THTMLTableColumns* object

DataLink

DataLink is an internal helper object that manages the link to the dataset.

property DataLink: THTTPDataLink;

Description Applications cannot use the *DataLink* property. It is exposed as a protected property so that descendant HTML producers can customize their response to data events.

See also *DataSet* property, *InternalDataSource* property, *THTTPDataLink* object

DataSet

DataSet is the dataset from which the data in the HTML table is taken.

property DataSet: TDataSet;

Description Set *DataSet* to specify where the *TDSTableProducer* should look for the data it displays in the HTML table. The individual field bindings specified by the *THTMLTableColumn* objects in the *Columns* property refer to fields in *DataSet*.

Specify the data binding of a *TDSTableProducer* using the *DataSet* property rather than through a *DataSource* property. *TDSTableProducer* objects do not use an external *TDataSource* object because they do not maintain record currency with other data objects in the application.

See also *DataSource* property, *InternalDataSource* property, *Columns* property, *GetDataSet* method, *SetDataSet* method, *THTMLTableColumn.FieldName* property

DataSource

DataSource represents the link to the *DataSet* of the *TDSTableProducer*.

property DataSource: TDataSource;

Description Do not use the *DataSource* property. The *DataSource* of a *TDSTableProducer* object is an internally generated *TDataSource* object that is used for internal interfaces that do not work directly with a *TDataSet* object.

The *InternalDataSource* property provides read-only access to the data source represented by *DataSource*.

See also *DataSet* property, *GetDataSource* method, *InternalDataSource* property, *SetDataSource* method

Editor

Editor is a helper object that is used at design time to write changes made in the property editor to the *TDSTableProducer* object.

```
property Editor: TDSTableProducerEditor;
```

Description Descendants of *TDSTableProducer* set the *Editor* property to the *TDSTableProducerEditor* object that is designed to connect their implementation with a common property editor. *Editor* allows a single property editor to represent the properties of different table producers.

See also *TDSTableProducerEditor* object

Footer

Footer is a set of HTML commands that appear in the HTML document following the HTML table that represents the *DataSet*.

```
property Footer: TStrings;
```

Description Set *Footer* to embed the representation of *DataSet* in a larger document. The *Content* method of *TDSTableProducer* returns any *Header* commands, followed by the HTML table that represents the *DataSet*, followed by any *Footer* commands.

See also *Content* method, *DataSet* property, *Header* property, *TStrings* object

Header

Header is a set of HTML commands that appear in the HTML document preceding the HTML table that represents the *DataSet*.

```
property Header: TStrings;
```

Description Set *Header* to embed the representation of *DataSet* in a larger document. The *Content* method of *TDSTableProducer* returns any *Header* commands, followed by the HTML table that represents the *DataSet*, followed by any *Footer* commands.

See also *Content* method, *DataSet* property, *Footer* property, *TStrings* object

InternalDataSource

InternalDataSource is the *DataSource* object generated internally by *TDSTableProducer* to represent the link to the *DataSet*.

```
property InternalDataSource: TDataSource;
```

Description Descendants of *TDSTableProducer* read *InternalDataSource* to access the properties and methods of the data source. *TDSTableProducer* does not use an external *TDataSource* object because it does not maintain record currency with other data objects in the application.

See also *DataSet* property, *TDataSource* object

MaxRows

MaxRows is the maximum number of rows that appear in the HTML table that represents the *DataSet*.

```
property MaxRows: Integer;
```

Description Set *MaxRows* to change the upper limit on the size of the HTML table produced to represent the *DataSet*. *MaxRows* is initialized to 20.

If *DataSet* contains more than *MaxRows* records, only the first *MaxRows* records appear in the HTML commands produced by *Content*. If *DataSet* contains fewer than *MaxRows* records, the HTML table produced by *TDSTableProducer* will contain only enough rows to display all the records in *DataSet*.

See also *Content* method, *DataSet* property

RowAttributes

RowAttributes specifies the display attributes of all the rows in the HTML table produced by *TDSTableProducer*.

```
property RowAttributes: THTMLTableRowAttributes;
```

Description Set *RowAttributes* to a *THTMLTableRowAttributes* object that describes the display attributes of the rows in the HTML table. Setting *RowAttributes* calls the *Assign* method of the *THTMLTableRowAttributes* object stored internally by *TDSTableProducer*. Thus, the memory for any row attributes object assigned to the *RowAttributes* property is never owned or freed by *TDSTableProducer*.

Note Not all the properties of the row attributes object are copied when one row attributes object is assigned to another. To ensure that all the attributes represented by *RowAttributes* are set appropriately, read *RowAttributes* and set the individual properties of the *THTMLTableRowAttributes* object.

See also *TableAttributes* property, *THTMLTableColumn.Title* property, *THTMLTableElementAttributes.AssignTo* method

TableAttributes

TableAttributes specifies the display attributes that pertain to the entire HTML table produced by *TDSTableProducer*.

```
property TableAttributes: THTMLTableAttributes;
```

Description Set *TableAttributes* to a *THTMLTableAttributes* object that describes the display attributes for the HTML TABLE tag. Setting *TableAttributes* calls the *Assign* method of the *THTMLTableAttributes* object stored internally by *TDSTableProducer*. Thus, the memory for any table attributes object assigned to the *TableAttributes* property is never owned or freed by *TDSTableProducer*.

Note Not all the properties of the table attributes object are copied when one table attributes object is assigned to another. To ensure that all the attributes represented by *TableAttributes* are set appropriately, read *TableAttributes* and set the individual properties of the *THTMLTableAttributes* object.

See also *RowAttributes* property, *THTMLTableAttributes.AssignTo* method, *THTMLTableColumn.Title* property

Events

The following table lists the events of *TDSTableProducer*:

Event	Visibility	Documented in
OnCreateContent	protected	TDSTableProducer
OnFormatCell	protected	TDSTableProducer
OnGetTableCaption	protected	TDSTableProducer

OnCreateContent

OnCreateContent occurs in descendants of *TDSTableProducer* when the *Content* method needs to generate the HTML table image.

```
type TCreateContentEvent = procedure (Sender: TObject; var Continue: Boolean) of object;

property OnCreateContent: TCreateContentEvent;
```

Description Write an *OnCreateContent* event handler to conditionally prevent the *Content* method from producing any HTML commands. Set the *Continue* parameter to *False* to make *Content* return an empty string.

See also *Content* method, *DoCreateContent* method

OnFormatCell

OnFormatCell occurs when the HTML for the cell specified by the *CellRow* and *CellColumn* parameters is about to be generated.

```
type THTMLFormatCellEvent = procedure (Sender: TObject; CellRow, CellColumn: Integer;
  var BgColor: THTMLBgColor; var Align: THTMLAlign; var VAlign: THTMLVAlign;
  var CustomAttrs, CellData: string) of object;

property OnFormatCell: THTMLFormatCellEventl;
```

Description Write an *OnFormatCell* event handler to change the appearance or contents of individual cells in the HTML table from the default values produced by

TDSTableProducer. *CellRow* and *CellColumn* specify the cell that is about to be generated, where the table header is *CellRow* 0, and the first column is *CellColumn* 0.

BgColor, *Align*, *VAlign*, and *CustomAttrs* represent the default display properties for the cell, taken from the *THTMLTableColumn* object for the column in which the cell appears. *CellData* is the text that appears in the cell, taken from the field object for the column. The *OnFormatCell* event handler can change any or all of these values to specify the way the cell should appear in the final HTML table.

See also *Columns* property, *Content* method, *DoFormatCell* method, *TField.DisplayText* property, *THTMLTableColumn.Align* property, *THTMLTableColumn.BgColor* property, *THTMLTableColumn.Custom* property, *THTMLTableColumn.VAlign* property

OnGetTableCaption

OnGetTableCaption occurs when the HTML table producer is about to generate the HTML for the table caption.

```
type THTMLGetTableCaptionEvent = procedure (Sender: TObject; var Caption: string;
  var Alignment: THTMLCaptionAlignment) of object;

property OnGetTableCaption: THTMLGetTableCaptionEvent;
```

Description Write an *OnGetTableCaption* event handler to change the value or placement of the HTML table caption. The *Caption* and *Alignment* parameters are originally set to the values of the *Caption* and *CaptionAlignment* properties, respectively. The *OnGetTableCaption* event handler can change these values to alter the caption for the HTML table returned by the *Content* method.

See also *Caption* property, *CaptionAlignment* property, *Content* method, *DoGetCaption* method

Methods

The following table lists the methods of *TDSTableProducer*:

Method	Visibility	Documented in
AcquireLayoutLock	protected	TDSTableProducer
Assign	public	TPersistent
BeginLayout	protected	TDSTableProducer
BeginUpdate	public	TDSTableProducer
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject

Method	Visibility	Documented in
CleanupInstance	public	TObject
Content	public	TCustomContentProducer
ContentFromStream	public	TCustomContentProducer
ContentFromString	public	TCustomContentProducer
Create	public	TDSTableProducer
DefaultHandler	public	TObject
DefineFieldMap	protected	TDSTableProducer
Destroy	public	TDSTableProducer
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
DoCreateContent	protected	TDSTableProducer
DoFormatCell	protected	TDSTableProducer
DoGetCaption	protected	TDSTableProducer
EndLayout	protected	TDSTableProducer
EndUpdate	public	TDSTableProducer
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetDataSet	protected	TDSTableProducer
GetDataSource	protected	TDSTableProducer
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
LayoutChanged	protected	TDSTableProducer
LinkActive	protected	TDSTableProducer
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Notification	protected	TDSTableProducer
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Method	Visibility	Documented in
SetColumns	protected	TDSTableProducer
SetDataSet	protected	TDSTableProducer
SetDataSource	protected	TDSTableProducer
StoreColumns	protected	TDSTableProducer

AcquireLayoutLock

AcquireLayoutLock calls *BeginLayout* when the HTML table is not currently being laid out.

```
function AcquireLayoutLock: Boolean;
```

Description Applications cannot call this protected method. *AcquireLayoutLock* is called before recomputing *Columns* when the fields in the dataset change or when the *Columns* property changes. *AquireLayoutLock* returns *True* if the HTML table is not already being laid out, to indicate that the computation of *Columns* should proceed. *AquireLayoutLock* prevents the computation of *Columns* from looping endlessly in response to the changes that occur during the computation.

See also *BeginLayout* method, *Columns* property, *LayoutChanged* method

BeginLayout

BeginLayout increments a counter that indicates the number of times the computation of the column order was begun but not completed.

```
procedure BeginLayout;
```

Description Applications cannot call this protected method. *TDSTableProducer* calls *BeginLayout* internally before making changes to the *Columns* property or any of the individual *THTMLTableColumn* objects in *Columns*. Once the change is complete, *TDSTableProducer* calls *EndLayout*.

When all *BeginLayout* calls have been matched by a corresponding call to *EndLayout*, the field mapping maintained by *DataLink* for the fields in the dataset is updated to match both the field bindings of the *Columns* property and the fields available in the dataset. The *Columns* property is updated in turn to match the field bindings in *DataLink*. This ensures that the *Columns* property matches both the specified column bindings and the fields available in the dataset.

See also *AcquireLayoutLock* method, *BeginUpdate* property, *Columns* property, *DataLink* property, *DefineFieldMap* method, *EndLayout* method

BeginUpdate

BeginUpdate increments a counter that indicates the number of changes to the contents of the HTML table that have been initiated but not completed.

procedure BeginUpdate;

Description Applications should not call *BeginUpdate*. *BeginUpdate* is called internally before changes that affect the contents of the HTML table. Once a change is complete, *EndUpdate* is called. *BeginUpdate* and *EndUpdate* allow descendants of *TDSTableProducer* to wait until changes to the dataset are complete before fetching records for the HTML image of the table.

See also *BeginLayout* property, *DataSet* property, *EndUpdate* method

Create

Create creates an instance of *TDSTableProducer*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to create a table producer from within the *OnAction* event handler of a *TWebActionItem* that needs to produce an HTML image of a dataset in response to an HTTP request message.

After calling the inherited constructor, *Create* allocates all the helper objects used by *TDSTableProducer* and initializes the *MaxRows* property to 20.

See also *MaxRows* property, *TComponent.Create* method

DefineFieldMap

DefineFieldMap builds the *Fields* array in *DataLink*.

procedure DefineFieldMap;

Description Applications cannot call this protected method. *DefineFieldMap* rebuilds the field mapping for the dataset that is maintained by *DataLink*. This mapping reflects both the column bindings of the *Columns* property and the fields available in the dataset. After calling *DefineFieldMap*, the *Columns* property should be updated to match the field bindings in *DataLink*. This ensures that the *Columns* property reflects any changes imposed by the dataset.

See also *Columns* property, *DataSet* property, *EndLayout* method, *THTTPDataLink.Fields* property

Destroy

Destroy destroys an instance of *TDSTableProducer*.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the *TDSTableProducer* object is not already freed, and only then calls *Destroy*.

Destroy frees all the helper objects used by *TDSTableProducer*.

See also *Free* method

DoCreateContent

DoCreateContent indicates whether the *Content* method should produce a string of HTML commands.

```
function DoCreateContent: Boolean;
```

Description Applications cannot call this protected method. It is called internally by the *Content* method to allow the *OnCreateContent* event handler to block the creation of any HTML commands. If *DoCreateContent* returns *True*, *Content* assembles a string of HTML to describe the dataset. If *DoCreateContent* returns *False*, *Content* returns an empty string.

DoCreateContent calls the *OnCreateContent* event handler if one is assigned. If there is no *OnCreateContent* event handler, *DoCreateContent* returns *True*. Descendants of *TDSTableProducer* can override *DoCreateContent* to add additional restrictions on when the HTML describing *DataSet* should be produced, or to block the *OnCreateContent* event.

See also *Content* method, *DataSet* property, *OnCreateContent* event

DoFormatCell

DoFormatCell calls the *OnFormatCell* event handler to allow changes to the contents or display attributes of an individual cell in the HTML table.

```
procedure DoFormatCell(CellRow, CellColumn: Integer; var BgColor: THTMLBgColor;
  var Align:THTMLAlign; var VAlign: THTMLVAlign; var CustomAttrs, CellData: string); dynamic;
```

Description Applications cannot call this protected method. It is called internally by the *Content* method to allow the *OnFormatCell* event handler to customize the contents or display attributes of any cell in the HTML table. *CellRow* and *CellColumn* indicate the cell about to be rendered as HTML, where *CellRow* for the table header is 0, and *CellColumn* for the first column is 0. *BgColor*, *Align*, *VAlign*, and *CustomAttrs* are the display attributes that can be changed. *CellData* is the text that goes in the cell.

Descendants of *TDSTableProducer* can override *DoFormatCell* to make additional changes to the cells as they are rendered or to block the *OnFormatCell* event.

See also *Content* method, *OnFormatCell* event

DoGetCaption

DoGetCaption calls the *OnGetCaption* event handler to allow changes to the value or placement of the table caption when it is rendered.

```
procedure DoGetCaption(var TableCaption: string; var CaptionAlign: THTMLCaptionAlignment);
  dynamic;
```

Description Applications cannot call this protected method. It is called internally by the *Content* method to allow the *OnGetCaption* event handler to change the

Caption and *CaptionAlignment* properties when the table caption is rendered as HTML.

Descendants of *TDSTableProducer* can override *DoCreateCaption* to make additional changes when the caption is rendered or to block the *OnCreateCaption* event.

See also *Caption* property, *CaptionAlignment* property, *Content* method, *OnCreateCaption* event

EndLayout

EndLayout decrements a counter that indicates the number of times the computation of the column order has been begun but not completed.

```
procedure EndLayout;
```

Description Applications cannot call this protected method. *TDSTableProducer* objects call *EndLayout* internally after making changes to the *Columns* property or when changes are made to the fields in *DataSet*.

Each call to *EndLayout* should match a corresponding call to *BeginLayout*. If all *BeginLayout* calls have been matched by an *EndLayout* call, *EndLayout* calls *DefineFieldMap* to regenerate the field mapping for the dataset maintained by *DataLink*. The *Columns* property is then updated to match the field map in *DataLink*. This ensures that the *Columns* property matches both the specified column bindings and the fields available in the dataset.

See also *BeginLayout* property, *Columns* property, *DataLink* property, *DefineFieldMap* method, *EndUpdate* method

EndUpdate

EndUpdate decrements a counter that indicates the number of changes to the contents of the HTML table.

```
procedure EndUpdate;
```

Description Applications should not call *EndUpdate*. *EndUpdate* is called internally after changes to the column bindings or *DataSet* of the *TDSTableProducer*. *BeginUpdate* and *EndUpdate* allow descendants of *TDSTableProducer* to wait until changes to the dataset are complete before fetching records for the HTML image of the table.

See also *BeginUpdate* property, *Columns* property, *DataSet* property, *EndLayout* method

GetDataSet

GetDataSet is the abstract prototype of a method to access the *DataSet* property.

```
function GetDataSet: TDataSet; virtual; abstract;
```

Description Descendants of *TDSTableProducer* implement the *GetDataSet* method to access their specific implementation of the *DataSet* property. Descendants of *TDSTableProducer* restrict the *DataSet* property to different classes of *TDataSet* objects.

See also *DataSet* property, *SetDataSet* method, *TDataSet* object

GetDataSource

GetDataSource is the protected property accessor for the *DataSource* property.

```
function GetDataSource: TDataSource;
```

Description *GetDataSource* is exposed as a protected method so that descendants of *TDSTableProducer* can override the implementation of the *DataSource* property.

The *DataSource* property for *TDSTableProducer* is an internally generated *TDataSource* object, not an external component that can be shared with other objects. It is surfaced only for internal APIs that require a *TDataSource* object rather than working directly with *TDataSet*.

See also *DataSource* property, *SetDataSource* method

LayoutChanged

LayoutChanged causes the *Columns* property to be recomputed when the dataset changes or changes are made to individual column objects.

```
procedure LayoutChanged;
```

Description Applications cannot call this protected method. It is called when the *Columns* property changes or the fields in the dataset change. *LayoutChanged* recomputes the *Columns* property after guarding against recursion from the changes to the *Columns* property that arise in the computation.

LayoutChanged calls *AcquireLayoutLock* to initiate the changes to *Columns*. If *AcquireLayoutLock* does not indicate that the call to *LayoutChanged* is recursive, *LayoutChanged* calls *EndLayout* to perform the actual computation.

See also *AcquireLayoutLock* method, *Columns* property, *EndLayout* method

LinkActive

LinkActive recomputes the *Columns* property when the dataset becomes active.

```
procedure LinkActive(Value: Boolean);
```

Description Applications cannot call this protected method. *LinkActive* is called when the *DataSource* property changes or when the dataset becomes active. It recomputes *Columns* to reflect any changes that may have occurred in the fields of the dataset when it was not active.

See also *Columns* property, *DataSource* property, *LayoutChanged* method

Notification

Notification adjusts the properties of *TDSTableProducer* when its *DataSource* or one of the field objects referenced by a column are destroyed.

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

Description *Notification* overrides the inherited method to respond to messages that the *DataSource* or one of its field objects is about to be destroyed. *Notification* sets the *DataSource* property to **nil**, or unbinds the column that references the field component.

See also *DataSource* property, *Columns* property

SetColumns

SetColumns sets the *Columns* property.

```
procedure SetColumns(Value: THTMLTableColumns);
```

Description *SetColumns* is the protected method used to set the *Columns* property. *SetColumns* calls the *Assign* method of the *THTMLTableColumns* object stored internally by *TDSTableProducer*. Thus, the memory for any columns object assigned to the *Columns* property is never owned or freed by *TDSTableProducer*.

See also *Columns* property, *TCollection.Assign* method, *THTMLTableColumn.AssignTo* method

SetDataSet

SetDataSet is the abstract prototype of a method to set the *DataSet* property.

```
procedure SetDataSet(ADataSet: TDataSet); virtual; abstract;
```

Description Descendants of *TDSTableProducer* override *SetDataSet* to ensure that the *DataSet* property can only be set to an appropriate descendant of *TDataSet*.

See also *DataSet* property, *GetDataSet* method

SetDataSource

SetDataSource sets the *DataSource* property.

```
procedure SetDataSource(Value: TDataSource);
```

Description *SetDataSource* is exposed as a protected method so that descendants of *TDSTableProducer* can override the implementation of the *DataSource* property.

The *DataSource* property for *TDSTableProducer* is an internally generated *TDataSource* object, not an external component that can be shared with other objects. It is surfaced only for internal API calls that require a *TDataSource* object rather than working directly with *TDataSet*.

See also *DataSource* property, *GetDataSource* method

StoreColumns

StoreColumns indicates whether the *Columns* property is stored when the *TDSTableProducer* is saved.

```
function StoreColumns: Boolean;
```

Description *StoreColumns* causes the *Columns* property to be stored only when the *Columns* property has a *State* property of *csCustom*. The *State* property for *Columns* indicates whether the columns represent default display attributes for all the fields in the dataset, or if the display attributes or field bindings of the columns have been customized programmatically or by the Columns editor.

See also *Columns* property, *THTMLTableColumns.State* property

TEdit

Unit: StdCtrls | Palette: Standard

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomEdit • TEdit

TEdit is a wrapper for a Windows single-line edit control.

Description Use a *TEdit* object to put a standard Windows edit control on a form. Edit controls are used to retrieve text that users type. Edit controls can also display text to the user.

When only displaying text to the user, choose an edit control to allow users to select text and copy it to the Clipboard. Choose a label object if the selection capabilities of an edit control are not needed.

TEdit implements the generic behavior introduced in *TCustomEdit*. *TEdit* publishes many of the properties inherited from *TCustomEdit*, but does not introduce any new behavior. For specialized edit controls, use other descendant classes of *TCustomEdit* or derive from it.

See also *TDBEdit* object, *TDBMemo* object, *TInplaceEdit* object, *TLabel* object, *TMaskEdit* object, *TMemo* object, *TRichEdit* object

Properties

The following table lists the properties of *TEdit*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
AutoSelect	published	True	TCustomEdit
AutoSize	published	True	TCustomEdit
BorderStyle	published	bsSingle	TCustomEdit
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
CharCase	published	ecNormal	TCustomEdit
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
HideSelection	published	True	TCustomEdit
Hint	published		TControl
ImeMode	published	imDontCare	TWinControl
ImeName	published		TWinControl
Left	published		TControl
MaxLength	published	0	TCustomEdit
Modified	public		TCustomEdit
Name	published		TControl
OEMConvert	published	False	TCustomEdit
Owner	public, read-only		TComponent

Property	Visibility	Default	Documented in
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PasswordChar	published	#0	TCustomEdit
PopupMenu	published		TControl
ReadOnly	published	False	TCustomEdit
SelLength	public		TCustomEdit
SelStart	public		TCustomEdit
SelText	public		TCustomEdit
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent
Text	published		TControl
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Events

The following table lists the events of *TEdit*:

Event	Visibility	Documented in
OnChange	published	TCustomEdit
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl

Event	Visibility	Documented in
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TEdit:*

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TCustomEdit
ClearSelection	public	TCustomEdit
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
CopyToClipboard	public	TCustomEdit
Create	public	TCustomEdit
CreateParented	public	TWinControl
CutToClipboard	public	TCustomEdit
DefaultHandler	public	TObject
Destroy	public	TWinControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent

Method	Visibility	Documented in
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetSelTextBuf	public	TCustomEdit
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
PasteFromClipboard	public	TCustomEdit
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SelectAll	public	TCustomEdit
SendToBack	public	TControl
SetBounds	public	TWinControl

Method	Visibility	Documented in
SetFocus	public	TWinControl
SetSelTextBuf	public	TCustomEdit
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

TField

Unit: DB

TObject • TPersistent • TComponent • TField

TField is the common ancestor of all the field components in a dataset.

Description

TField encapsulates the fundamental behavior common to all field components. It introduces the properties, events, and methods that are used to:

- Change the value of a field in a dataset.
- Convert the value of a field from one data type to another.
- Validate data that the user enters for a field.
- Define how the data in the field appears as it is displayed or edited.
- Calculate the value of a field from code written in the *OnCalcFields* event of the dataset.
- Look up the field's value from another dataset.

Do not create instances of *TField*. *TField* descendants are created automatically each time a dataset is activated. These descendants can be dynamic (the default) or persistent. Dynamic field components reflect the actual columns of the underlying physical table at the time a dataset is opened. Persistent field components are created at design time using the *Fields* editor, which specifies the fields in the dataset, their properties, and their ordering.

Creating persistent field components guarantees that each time an application runs, it uses and displays the same columns, in the same order, even if the physical structure of the underlying database changes. If a column on which a persistent field component is based is deleted or changed, Delphi generates an exception rather than opening the table against a nonexistent column or mismatched data. If this happens, remove the field component for the nonexistent field using the Fields editor.

A field in a dataset is always one of the following *TField* descendants:

- *TStringField*
- *TSmallintField*
- *TFloatField*
- *TCurrencyField*
- *TBooleanField*
- *TDateField*
- *TBlobField*
- *TIntegerField*
- *TWordField*
- *TAutoIncField*
- *TBCDField*
- *TDateTimeField*
- *TTimeField*
- *TBytesField*
- *TVarBytesField*
- *TGraphicField*
- *TMemoField*

See also *TAutoIncField* object, *TBCDField* object, *TBlobField* object, *TBooleanField* object, *TBytesField* object, *TCurrencyField object*, *TDataSet.Fields* property, *TDataSet.OnCalcFields* event, *TDateField* object, *TDateTimeField* object, *TFieldDef* object, *TFieldDefs* object, *TFloatField* object, *TGraphicField* object, *TIntegerField* object, *TMemoField* object, *TNumericField* object, *TSmallintField* object, *TStringField* object, *TVarBytes Field* object, *TWordField* object

Properties

The following table lists the properties of *TField*:

Property	Visibility	Default	Documented in
Alignment	published	taLeftJustify	TField
AsBoolean	public		TField
AsCurrency	public		TField
AsDateTime	public		TField
AsFloat	public		TField
AsInteger	public		TField
AsString	public		TField
AsVariant	public		TField
AttributeSet	public		TField
Calculated	public	False	TField
CanModify	public, read-only		TField
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ConstraintErrorMessage	published		TField
CurValue	public, read-only		TField
CustomConstraint	published		TField
DataSet	public		TField
DataSize	public, read-only		TField

Property	Visibility	Default	Documented in
DataType	public, read-only		TField
DefaultExpression	published		TField
DesignInfo	public		TComponent
DisplayLabel	published		TField
DisplayName	public, read-only		TField
DisplayText	public, read-only		TField
DisplayWidth	published		TField
EditMask	public		TField
EditMaskPtr	public, read-only		TField
FieldKind	published	fkData	TField
FieldName	published		TField
FieldNo	public, read-only		TField
HasConstraints	published, read-only		TField
ImportedConstraint	published		TField
Index	published		TField
IsIndexField	public, read-only		TField
IsNull	public, read-only		TField
KeyFields	published		TField
Lookup	public		TField
LookupCache	published	False	TField
LookupDataSet	published		TField
LookupKeyFields	published		TField
LookupList	public, read-only		TField
LookupResultField	published		TField
Name	published		TComponent
NewValue	public		TField
Offset	public, read-only		TField
OldValue	public, read-only		TField
Origin	published		TField
Owner	public, read-only		TComponent
ReadOnly	published	False	TField
Required	published	False	TField
Size	public		TField
Tag	published	0	TComponent
Text	public		TField
ValidChars	public		TField
Value	public		TField
VCLComObject	public		TComponent
Visible	published	True	TField

Alignment

Alignment determines how the field's data is displayed within a data-aware control.

property Alignment: TAlignment;

Description Use *Alignment* to specify the position of the field's data within data-aware controls. These are the possible values:

Value	Meaning
taLeftJustify	Align data to the left side of the control
taCenter	Center data horizontally in the control
taRightJustify	Align data to the right side of the control

See also *DisplayLabel* property, *DisplayName* property, *DisplayText* property, *DisplayWidth* property, *TDBEdit* object

AsBoolean

AsBoolean introduces a property to convert between the field's native type and a Boolean value.

property AsBoolean: Boolean;

Description *AsBoolean* provides a property to read the value of the field's data into a Boolean, or to assign a Boolean value to the contents of the field. *TField* does not support Boolean values, and raises an exception whenever an attempt is made to get or set the *AsBoolean* property.

Descendants of *TField* that represent Boolean fields or that support conversions between the field's *Value* property and a Boolean value, override *AsBoolean* to read and write the value of the field as a Boolean value.

See also *DataType* property, *Value* property

AsCurrency

AsCurrency introduces a property to convert between the field's native type and a Currency value.

property AsCurrency: Currency;

Description *AsCurrency* provides a property to read the value of the field's data into an object or variable of type Currency, or to assign a Currency value to the contents of the field. *TField* does not support Currency values, and raises an exception whenever an attempt is made to get or set the *AsCurrency* property.

Descendants of *TField* that represent currency fields or that support conversions between the field's *Value* property and a Currency value,

override *AsCurrency* to read and write the value of the field as a Currency value.

See also *AsFloat* property, *DataType* property, *Value* property

AsDateTime

AsDateTime introduces a property to convert between the field's native type and a *TDateTime* value.

```
property AsDateTime: TDateTime;
```

Description *AsDateTime* provides a property to read the value of the field's data into an object or variable of type *TDateTime*, or to assign a *TDateTime* value to the contents of the field. *TField* does not support *TDateTime* values, and raises an exception whenever an attempt is made to get or set the *AsDateTime* property.

Descendants of *TField* that represent datetime (timestamp) fields or that support conversions between the field's *Value* property and a *TDateTime* value, override *AsDateTime* to read and write the value of the field as a *TDateTime* value.

See also *DataType* property, *TDataTime* type, *Value* property

AsFloat

AsFloat introduces a property to convert between the field's native type and a Double value.

```
property AsFloat: Double;
```

Description *AsFloat* provides a property to read the value of the field's data into a Double, or to assign a Double value to the contents of the field. *TField* does not support floating point values, and raises an exception whenever an attempt is made to get or set the *AsFloat* property.

Descendants of *TField* that represent floating point numeric fields or that support conversions between the field's *Value* property and a Double value, override *AsFloat* to read and write the value of the field as a Double value.

See also *AsCurrency* property, *DataType* property, *Value* property

AsInteger

AsInteger introduces a property to convert between the field's native type and a 32-bit integer.

```
property AsInteger: Integer;
```

Description *AsInteger* provides a property to read the value of the field's data into an integer, or to assign an integer value to the contents of the field. *TField* does

not support integer values, and raises an exception whenever an attempt is made to get or set the *AsInteger* property.

Descendants of *TField* that represent integer fields or that support conversions between the field's *Value* property and an integer value, override *AsInteger* to read and write the value of the field as an integer value.

See also *AsFloat* property, *DataType* property, *Value* property

AsString

AsString is the abbreviated class name of the field component.

```
property AsString: string;
```

Description *Reading the AsString* property returns the abbreviated class name of the field component enclosed in parentheses. The abbreviated class name is built by stripping the leading 'T' from the class name and removing the trailing word 'FIELD', except in the case of *TField*, where that would leave an empty string. For *TField*, the abbreviated class name is 'FIELD'. Trying to set the *AsString* property raises an exception.

Descendants of *TField* that represent string fields or that support conversions between the field's *Value* property and a string, override *AsString* to read and write the value of the field as a different, more meaningful, string.

See also *DataType* property, *Value* property

AsVariant

AsVariant is the *Value* of the field expressed as type Variant.

```
property AsVariant: Variant;
```

Description *AsVariant* provides a property to read the value of the field's data into a Variant, or to assign a Variant value to the contents of the field. *TField* does not support *Variant* values, and raises an exception when an attempt is made to read the *AsVariant* property. *TField* also raises an exception when an attempt is made to set the *AsVariant* property, unless the Variant is of type *varNull*, in which case *TField* calls the *Clear* method.

Descendants of *TField* implement the *AsVariant* method to get or set the value of the field as a Variant.

See also *Clear* method, *Value* property

AttributeSet

AttributeSet is the name of the attribute set in the data dictionary to apply to a field component.

```
property AttributeSet: string;
```

Description An attribute set defines the display and format properties of a field component that are displayed in a data-aware control at design time. It also defines any constraint and default properties of the field component.

Attribute sets are stored in the data dictionary by name so that at design time developers can create and apply the same set of attributes to different field components in one or more datasets. When an attribute set is assigned to a field component at design time and that field component is owned by a *TQuery* or *TStoredProc* dataset, *AttributeSet* contains the name of the assigned attribute set.

See also *TDataSet* object, *TQuery* object, *TStoredProc* object

Calculated

Calculated determines whether the value of the field is calculated by the *OnCalcFields* event handler of a dataset.

```
property Calculated: Boolean;
```

Description Use *Calculated* to determine if the field is a calculated field. When *Calculated* is *True* and the *OnCalcFields* event occurs, the *OnCalcFields* event handler updates the value of the field. The default value is *False*.

The value of a *Calculated* field is not stored in or retrieved from the physical tables underlying a dataset. Instead, calculated fields are calculated for each record in the table by the dataset's *OnCalcFields* event handler, which typically uses expressions involving values from other fields in the record to generate a value for each calculated field.

For example, a table might have non-calculated fields for *Quantity* and *UnitPrice*, and a calculated field for *ExtendedPrice*, which would be calculated by multiplying the values of the *Quantity* and *UnitPrice* fields in an *OnCalcFields* event handler.

Do not change the setting of *Calculated* in application code. *Calculated* is automatically set to *True* by the *Fields* editor at design time if a field's value is specified as calculated.

Note Do not confuse the *Calculated* property with the *InternalCalcField* property of a field definition. The value of a calculated field is computed in the *OnCalcFields* event handler. The value of an internally calculated field is computed by the SQL server or the Borland Database engine.

See also *TDataSet.OnCalcFields* event, *TFieldDef.InternalCalcField* property

CanModify

CanModify specifies if a field can be modified.

```
property CanModify: Boolean;
```

Description Check *CanModify* to determine if a field's data can be edited. *CanModify* is *True* only if the *ReadOnly* property is *False*, and the dataset for this field component allows write access to the corresponding database field.

See also *DataSet* property, *ReadOnly* property

ConstraintErrorMessage

ConstraintErrorMessage is a customized error message that appears when the user tries to set a value that violates the data constraints for the field.

```
property ConstraintErrorMessage: string;
```

Description Set *ConstraintErrorMessage* to specify the message that appears when the user-supplied value for the field does not meet the constraints imposed by the server or specified in the *CustomConstraint* property.

See also *CustomConstraint* property, *ImportedConstraint* property

CurValue

CurValue represents the current value of the field component including changes made by other users of the database.

```
property CurValue: Variant;
```

Description Use *CurValue* to examine the value of a field when a problem occurs in posting a value to the database. If the current field value is causing a problem, such as a key violation, when posting the value to the database, an event is generated to allow applications to respond to the problem. Client datasets generate an *OnReconcileError* event. On the server side, *TProvider* objects generate an *OnUpdateError* event. In the event handler, *NewValue* is the unposted value that caused the problem, *OldValue* is the value that was originally assigned to the field before any edits were made, and *CurValue* is the value that is currently assigned to the field. *CurValue* may differ from *OldValue* if another user changed the value of the field after *OldValue* was read.

Note *CurValue* is only supported when the dataset is a *TClientDataSet* or the source dataset for *TProvider*.

See also *NewValue* property, *OldValue* property, *TProvider.OnUpdateError* event, *TClientDataSet.OnReconcileError* event

CustomConstraint

CustomConstraint is an SQL string the specifies any application-specific constraints imposed on the field's value.

```
property CustomConstraint: string;
```

Description Set *CustomConstraint* to limit the values that the user can enter into a field. *CustomConstraint* must be a valid SQL search expression such as

```
x > 0 and x < 100
```

The name used to refer to the value of the field can be any string that is not a reserved SQL keyword, as long as it is used consistently throughout the constraint expression.

Custom constraints are imposed in addition to any constraints to the field's value that come from the server. To see the constraints imposed by the server, read the *ImportedConstraint* property.

Note When setting the *CustomConstraint* property, be sure to set the *ConstraintErrorMessage* property as well, so that a message will appear to the user when a value is entered that violates the custom constraint.

See also *ConstraintErrorMessage* property, *ImportedConstraint* property

DataSet

DataSet identifies the dataset to which a field component belongs.

property DataSet: TDataSet;

Description Check the value of the *DataSet* property to determine the dataset that uses the field component to represent one of its fields. Applications should not directly assign the *DataSet* property of a field. It is assigned automatically when the field is created from a *TFieldDef* object.

See also *TFieldDef.CreateField* method

DataSize

DataSize is the amount of memory that is needed to store a field component's value.

property DataSize: Word;

Description Check *DataSize* to determine the number of bytes required to store a field component's value. Use *DataSize* to determine the required size of a buffer for working with the field's value in native format. For example, use *DataSize* to determine the buffer size needed by the *GetData* and *SetData* methods.

The value of *DataSize* for *TField* is 0. Most descendants of *TField* override this property to specify the size required by the specific field type they represent. BLOB fields, however, represent data of indeterminate size. They do not override the inherited method, and the value of *DataSize* for BLOB fields is 0. A value of zero should not be interpreted as meaning the field value requires no memory. A value of zero indicates that the memory required for the field is indeterminate.

See also *DataType property*, *GetData* method, *SetData* method

DataType

DataType identifies the data type of the *field component*.

```
type TFieldType = (ftUnknown, ftString, ftSmallint, ftInteger, ftWord, ftBoolean, ftFloat, ftCurrency,
  ftBCD, ftDate, ftTime, ftDateTime, ftBytes, ftVarBytes, ftAutoInc, ftBlob, ftMemo, ftGraphic,
  ftFmtMemo, ftParadoxOle, ftDBaseOle, ftTypedBinary, ftCursor);

property DataType: TFieldType;
```

Description Read the *DataType* property to learn the type of the data the field contains. The *DataType* property is equal to one of the following values:

Value	Description
ftUnknown	Unknown or undetermined
ftString	Character or string field
ftSmallint	16-bit integer field
ftInteger	32-bit integer field
ftWord	16-bit unsigned integer field
ftBoolean	Boolean field
ftFloat	Floating-point numeric field
ftCurrency	Money field
ftBCD	Binary-coded decimal field
ftDate	Date field
ftTime	Time field
ftDateTime	Date and time field
ftBytes	Fixed number of bytes (binary storage)
ftVarBytes	Variable number of bytes (binary storage)
ftAutoInc	Auto-incrementing 32-bit integer counter field
ftBlob	Binary Large OBject field
ftMemo	Text memo field
ftGraphic	Bitmap field
ftFmtMemo	Formatted text memo field
ftParadoxOle	Paradox OLE field
ftDBaseOle	dBASE OLE field
ftTypedBinary	Typed binary field
ftCursor	Does not apply to field components.

See also *DataSize* property, *IsValidChar* method

DefaultExpression

DefaultExpression is an SQL expression that represents the default value that is assigned to the field if the user does not provide a value.

```
property DefaultExpression: string;
```

Description Set *DefaultExpression* to provide a default value for a field. *DefaultExpression* can be any valid SQL value expression that does not refer to field values. If the expression contains literals other than numeric values, they must appear in quotes. For example, a default value of noon for a time field would be

```
'12:00:00'
```

including the quotes around the literal value.

See also *Value* property

DisplayLabel

DisplayLabel contains the text to display in the corresponding column heading of a data grid.

property DisplayLabel: **string**;

Description Use *DisplayLabel* to assign column headings to a data grid. The column headings of the data grid use the *DisplayName* property of the field whose value they represent. Setting *DisplayLabel* changes *DisplayName* property from *FieldName* to the string specified as *DisplayLabel*.

See also *DisplayName property*, *FieldName* property, *TDBGrid* object

DisplayName

DisplayName returns the name of the field for display purposes.

property DisplayName: **string**;

Description Use *DisplayName* when displaying a string that represents the field to the user. For example, when raising an exception concerning the field, format an error string that includes the *DisplayName* to refer to the field. *DisplayName* is a read-only property. To change the value of *DisplayName*, use the *DisplayLabel* property.

If *DisplayLabel* is defined, *DisplayName* is the same as *DisplayLabel*. Otherwise, *DisplayName* is the same as the *FieldName* property.

See also *DisplayLabel* property, *FieldName* property

DisplayText

DisplayText is the string representation of a field's value to display in a data-aware control.

property DisplayText: **string**;

Description *DisplayText* is a read-only string representation of a field's value to display in a data-aware control. If an *OnGetText* event handler is assigned, *DisplayText* is the value returned in the *Text* parameter of the *OnGetText* event handler when the *DisplayText* parameter is *True*. Otherwise, *DisplayText* is the value of the *AsString* property.

DisplayText is the string representation of the field's *Value* property when it is not being edited. When the field is being edited, use the *Text* property.

See also *AsString* property, *OnGetText* event, *Text* property, *Value* property

DisplayWidth

DisplayWidth specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.

property DisplayWidth: Integer;

Description Set *DisplayWidth* to control how many characters are used to calculate the column width for the field when it is displayed by a *TDBLookupListBox* object or a descendant if *TCustomDBGrid*. The actual width of columns is determined by the average character width. If the data-aware control that displays the field is not using a fixed pitch font, *DisplayWidth* may indicate more or fewer characters than will fit in the column for any given string.

Note The *DisplayWidth* property has no effect if there is an explicit *Columns* value for the field in a *TDBGrid* object.

See also *DisplayText* property, *TCustomDBGrid* object, *TDBLookupListBox* object

EditMask

EditMask contains the mask that is used to restrict the data that can be entered into a data field.

property EditMask: **string**;

Description Use *EditMask* to restrict the characters the user can enter to valid characters and formats. If the user attempts to enter a character that is not valid, the character is not accepted. Validation using the *EditMask* property is performed on a character-by-character basis. Use an *OnValidate* event to validate the entire input, just before it is posted to the database.

A mask consists of three fields with semicolons separating the fields. The first part of the mask is the mask itself. The second part is the character that determines whether the literal characters of a mask are included as part of the data. The third part of the mask is the character used to represent unentered characters in the mask.

These are the special characters used in the first field of the mask:

Character	Meaning in mask
!	If a ! character appears in the mask, optional characters are represented in the *EditText* as leading blanks. If a ! character is not present, optional characters are represented in the *EditText* as trailing blanks.
>	If a > character appears in the mask, all characters that follow are in uppercase until the end of the mask or until a < character is encountered.

Character	Meaning in mask
<	If a < character appears in the mask, all characters that follow are in lowercase until the end of the mask or until a > character is encountered.
<>	If these two characters appear together in a mask, no case checking is done and the data is formatted with the case the user uses to enter the data.
\	The character that follows a \ character is a literal character. Use this character when you want to allow any of the mask special characters as a literal in the data.
L	The L character requires an alphabetic character only in this position. For the U.S., this is A-Z, a-z.
l	The l character permits only an alphabetic character in this position, but doesn't require it.
A	The A character requires an alphanumeric character only in this position. For the U.S., this is A-Z, a-z, 0-9.
a	The a character permits an alphanumeric character in this position, but doesn't require it.
C	The C character requires an arbitrary character in this position.
c	The c character permits an arbitrary character in this position, but doesn't require it.
0	The 0 character requires a numeric character only in this position.
9	The 9 character permits a numeric character in this position, but doesn't require it.
#	The # character permits a numeric character or a plus or minus sign in this position, but doesn't require it.
:	The : character is used to separate hours, minutes, and seconds in times. If the character that separates hours, minutes, and seconds is different in the regional settings of the Control Panel utility on your computer system, that character is used instead of :.
/	The / character is used to separate months, days, and years in dates. If the character that separates months, days, and years is different in the regional settings of the Control Panel utility on your computer system, that character is used instead of /.
;	The ; character is used to separate the three fields of the mask.
_	The _ character automatically inserts spaces into the text. When the user enters characters in the field, the cursor skips the _ character.

Any character that does not appear in the preceding table can appear in the first part of the mask as a literal character. Literal characters must be matched exactly when entering data for the field. They are inserted automatically, and the cursor skips over them during editing. The special mask characters can also appear as literal characters if preceded by a backslash character (\).

The second field of the mask is a single character that indicates whether literal characters from the mask should be included as part of the *Value* property of the field. For example, the mask for a telephone number with area code could be the following string:

```
(000)_000-0000;0;*
```

The 0 in the second field indicates that the *Value* property for the field would consist of the 10 digits that were entered, rather than the 14 characters that make up the telephone number as it appears to the user.

A 0 in the second field indicates that literals should not be included, any other character indicates that they should be included. The character to indicate whether literals should be included can be changed in the *EditMask* property editor, or programmatically by changing the *MaskNoSave* typed constant defined in the Mask unit.

The third field of the mask is the character that appears in the edit control for blanks (characters that have not been entered). By default, this is the same as the character that stands for literal spaces. The two characters appear the same in an edit window. However, when a user enters the data for the field, the cursor selects each blank character in turn, and skips over the space character.

Setting *EditMask* to an empty string removes the mask.

Note When working with multibyte character sets, such as Japanese Shift-JIS, each special mask character represents a single byte. To specify double-byte characters using the L, l, A, a, C, or c specifiers, the mask characters must be doubled as well. For example, LL would represent two single-byte alphabetic characters or a one double-byte character. Only single-byte literal characters are supported.

See also *DisplayText* property, *IsValidChar* method, *OnValidate* event, *Validate* method, *Value* property

EditMaskPtr

EditMaskPtr provides read-only access to the *EditMask* property.

```
property EditMaskPtr: string;
```

Description Use *EditMaskPtr* when reading the *EditMask* property. Because *EditMaskPtr* is a read-only property, *EditMaskPtr* guards against accidently changing the value of the mask in code that should only read it.

See also *EditMask* property

FieldKind

FieldKind indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.

```
type TFieldKind = (fkData, fkCalculated, fkLookup, fkInternalCalc);
```

```
property FieldKind: TFieldKind;
```

Description Use *FieldKind* to determine if a field is a data field, a calculated field, or a lookup field. The value of *FieldKind* can be changed programmatically, but in practice *FieldKind* is automatically set at design time when creating *Field* components with the *Fields* editor.

The *FieldKind* property can take one of the following values:

Value	Description
fkData	Field represents a physical field in a database table.
fkCalculated	Field is calculated in an *OnCalcFields* event handler.
fkLookup	Field is a lookup field.
fkInternalCalc	Field is calculated but values are stored in the dataset.

Note Fields calculated by SQL servers or the Borland Database Engine to display the results of a query that returns a live dataset have a *FieldKind* of *fkInternalCalc*, not *fkCalculated*. This is because the field values are stored in the dataset. Calculated fields in a client dataset are calculated in an *OnCalcFields* event handler, but stored internally in the dataset. They also have a *FieldKind* of *fkInternalCalc* instead of *fkCalculated*. Unlike regular calculated fields, these fields can be used in filter expressions. They can also be edited, but the changes are discarded. To prevent editing, set the *ReadOnly* property to *True*.

See also *Calculated* property, *Lookup* property, *ReadOnly* property

FieldName

FieldName is the name of the physical column in the underlying dataset to which a field component is bound.

```
property FieldName: string;
```

Description When creating a field component, the *TFieldDef* object uses *FieldName* to specify which field in underlying dataset the field component represents. *FieldName* is used when displaying references to the field to users, unless a *DisplayLabel* has been set. For calculated fields, supply a *FieldName* when defining the field. For non-calculated fields, an *EDatabaseError* exception occurs if a *FieldName* is not a column name in the physical table.

See also *DisplayLabel* property, *DisplayName* property, *EDatabaseError* object, *Origin* property, *TFieldDef.CreateField* method

FieldNo

FieldNo is the ordinal of the field component's actual column in the physical table underlying a dataset.

```
property FieldNo: Integer;
```

Description *Read FieldNo* to learn a field component's ordinal position in the table underlying the dataset, where 1 is the first field. Use the *FieldNo* property to specify the field only in direct API calls to the Borland Database Engine.

Note Don't confuse *FieldNo*, which is the field component's ordinal position in the physical database table, with *Index*, the field component's position in the dataset's *Fields* property.

See also *FieldName* property, *Index* property

HasConstraints

HasConstraints indicates whether the field values are constrained.

property HasConstraints: Boolean;

Description Read *HasConstraints* to determine whether the field component imposes custom constraints, imported constraints, or a default expression. *HasConstraints* is *True* if the *CustomConstraint*, *DefaultExpression*, or *ImportedConstraint* property is set. Otherwise, *HasConstraints* is *False*, indicating that the server or database engine are not imposing any constraints on field values.

Field values can still be limited when *HasConstraints* is *False*. For example, an edit mask or *OnValidate* event handler can constrain field values. However, these constraints are imposed by the Delphi application, not by the server or database engine.

See also *CustomConstraint* property, *DefaultExpression* property, *EditMask* property, *ImportedConstraint* property, *OnValidate* event

ImportedConstraint

ImportedConstraint is an SQL clause that indicates the constraints imposed by the server on the field's value.

property ImportedConstraint: **string**;

Description Read *ImportedConstraint* to determine the limits imposed by the server on what values are valid for the field. *ImportedConstraint* is an SQL search expression such as

```
Value > 0 and Value < 100
```

Do not change the value of *ImportedConstraint*. To add additional constraints on the field value, use the *CustomConstraint* property instead. Custom constraints are imposed in addition to the imported constraints. If the server constraints change, the value of *ImportedConstraint* will be changed but constraints introduced in the *CustomConstraint* property will persist.

Removing constraints from the *ImportedConstraint* property will not change the validity of field values that violate those constraints. Removing constraints will result in the constraints being checked by the server instead of locally. When constraints are checked locally, the error message supplied as the *ConstraintErrorMessage* property is displayed when violations are found, instead of an error message from the server.

See also *ConstraintErrorMessage* property, *CustomConstraint* property

Index

Index is the field component's index number into the *Fields* property of the dataset.

property Index: Integer;

Description Use the *Index* property to

- Find the position of the field in the dataset.
- Change the order of a field's position in the dataset by changing the value of *Index*. Changing the *Index* value affects the order in which fields are displayed in data grids, but not the position of the fields in the physical database tables.

Note The field order can also be changed by dragging and dropping fields in the *Fields* editor at design time.

See also *FieldNo* property, *TDataSet.Fields* property, *TDBGrid* component

IsIndexField

IsIndexField specifies whether a field is an index field.

property IsIndexField: Boolean;

Description Use the *IsIndexField* to determine if a field is an index field. If *IsIndex* is *True*, a field is one of the fields that makes up the index on the underlying database table. If *IsIndexField* is *False*, the field is not an index field. A field component must be an index field to serve as one of the field components for the *LookupKeyFields* in a lookup table.

See also *LookupKeyFields* property

IsNull

IsNull determines if the field has a value assigned to it.

property IsNull: Boolean;

Description Use *IsNull* to determine if the field contains a value. If *IsNull* is *True*, the field is blank. If *IsNull* is *False*, the field has a value.

See also *Clear* method

KeyFields

KeyFields identifies the field or fields in the dataset that must be matched in the lookup dataset when doing a lookup.

property KeyFields: **string**;

Description Set *KeyFields* only on lookup fields (fields where *FieldKind* is *fkLookup* and *Lookup* is *True*). *KeyFields* specifies the field or fields to match when a lookup occurs. To use more than one field, separate each field name with a semicolon. The specified fields must be of the same types as the corresponding fields in *LookupKeyFields*, or the lookup fails.

The values of the *KeyFields* in the *DataSet* are matched with the values of the *LookupKeyFields* in the *LookupDataSet* to determine a particular record of the lookup table. The value of the *LookupResultField* on that record becomes the *Value* property for the field component.

See also *DataSet* property, *FieldKind* property, *Lookup* property, *LookupDataSet* property, *LookupKeyFields* property, *LookupResultField* property, *Value* property

Lookup

Lookup determines whether the field is specified as a lookup field.

```
property Lookup: Boolean;
```

Description *Lookup* is provided for backward compatibility. To determine if a field is a lookup field, use the *FieldKind* property instead.

If the field is a lookup field, the *KeyFields* property indicates which fields in the dataset must match the *LookupKeyFields* in the *LookupDataSet*, to identify a record in the *LookupDataSet*. The value of the *LookupResultField* for that record becomes the *Value* of the field component.

See also *FieldKind* property, *KeyFields* property, *LookupCache* property, *LookupDataSet* property, *LookupKeyFields* property, *LookupResultField* property, *Value* property

LookupCache

LookupCache determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.

```
property LookupCache: Boolean;
```

Description Set *LookupCache* to *True* to cache the values of a lookup field when the *LookupDataSet* is unlikely to change and the number of distinct lookup values is small. Caching lookup values can speed performance, because the lookup values for every set of *LookupKeyFields* values are preloaded when the *DataSet* is opened. When the current record in the *DataSet* changes, the field object can locate its *Value* in the cache, rather than accessing the *LookupDataSet*. This performance improvement is especially dramatic if the *LookupDataSet* is on a network where access is slow.

If every record of *DataSet* has different values for *KeyFields*, the overhead of locating values in the cache can be greater than any performance benefit provided by the cache. The overhead of locating values in the cache increases with the number of distinct values that can be taken by *KeyFields*.

If *LookupDataSet* is volatile, caching lookup values can lead to inaccurate results. Call *RefreshLookupList* to update the values in the lookup cache. *RefreshLookupList* regenerates the *LookupList* property, which contains the value of the *LookupResultField* for every set of *LookupKeyFields* values.

When setting *LookupCache* at runtime, call *RefreshLookupList* to initialize the cache.

Note *LookupCache* is only meaningful if the value of *FieldKind* is *fkLookup*.

See also *DataSet* property, *FieldKind* property, *KeyFields* property, *LookupDataSet* property, *LookupKeyFields* property, *LookupList* property, *LookupResultField* property, *RefreshLookupList* method.

LookupDataSet

LookupDataSet identifies the dataset used to look up field values.

```
property LookupDataSet: TDataSet;
```

Description Use *LookupDataSet* to specify the dataset to use for looking up field values in a field with a *FieldKind* of *fkLookup*.

When a field component is a lookup field, the *Value* of the field component is the value of the *LookupResultField* in the *LookupDataSet*. The record in the *LookupDataSet* that contains the correct value is found by matching the *LookupKeyFields* in the *LookupDataSet* with the current values of the *KeyFields* in the *DataSet*.

See also *FieldKind* property, *KeyFields* property, *LookupKeyFields* property, *LookupResultField* property, *Value* property, *DataSet* property

LookupKeyFields

LookupKeyFields identifies the field or fields in the lookup dataset to match when doing a lookup.

```
property LookupKeyFields: string;
```

Description Only use *LookupKeyFields* on field components with a *FieldKind* of *fkLookup*. *LookupKeyFields* are the fields in the index of the *LookupDataSet* whose values must match the *KeyFields* in the field component's *DataSet*.

To specify more than one field, separate each field name with a semicolon. The fields specified in *LookupKeyFields* must be of the same types as the corresponding fields in *KeyFields*, or the lookup can't work.

Matching the values of the *LookupKeyFields* in the *LookupDataSet* with the values of the *KeyFields* in the *DataSet* determines a specific record in the *LookupDataSet*. The *Value* of the field component is the value of the *LookupResultField* in the indicated record.

See also *FieldKind* property, *KeyFields* property, *Lookup* property, *LookupDataSet* property, *LookupResultField* property, *Value* property, *DataSet* property

LookupList

LookupList is a cache of values from the *LookupDataSet* indexed by a set of values from the *KeyFields* property.

property LookupList: TLookupList;

Description Use *LookupList* to find the cached value assigned to the field for a given set of values on the fields specified by *KeyFields*. The value of *LookupList* is set when the dataset is open or the *RefreshLookupList* method is called. Applications can also set the *LookupList* programmatically to provide lookup values when the lookup dataset does not actually exist.

The *ValueOfKey* method of the *LookupList* returns the value of the *LookupResultField* for a set of values on the *LookupKeyFields*. This allows the value of the lookup field object to be computed as follows:

```
Value := LookupList.ValueOfKey(DataSet.FieldValues[KeyFields]);
```

Note *LookupList* is only meaningful when the value of *FieldKind* is *fkLookpup* and the *LookupCache* property is *True*.

See also *DataSet* property, *FieldKind* property, *KeyFields* property, *LookupCache* property, *LookupDataSet* property, *LookupKeyFields* property, *LookupResultField* property, *TDataSet.FieldValues* property, *TLookupList* object, *TLookupList.ValueOfKey* method

LookupResultField

LookupResultField identifies the field from the lookup dataset whose value becomes the *Value* property of the field component.

property LookupResultField: **string**;

Description Use *LookupResultField* to specify a single field in the *LookupDataSet* whose value becomes the value of the field component. The field component must have a *FieldKind* of *fkLookup*, and the *DataType* of the field component must be the same as the data type of the *LookupResultField*.

When a field component is a lookup field, the *Value* of the field component is the value of the *LookupResultField* in the *LookupDataSet*. The record in the *LookupDataSet* that contains the correct value is found by matching the *LookupKeyFields* in the *LookupDataSet* with the current values of the *KeyFields* in the *DataSet*.

See also *DataSet* property, *DataType* property, *FieldKind* property, *KeyFields* property, *LookupDataSet* property, *LookupKeyFields* property, *Value* property

NewValue

NewValue represents the current value of the field component including pending cached updates.

property NewValue: Variant;

Description Use *NewValue* to examine or change the current value to a field when in the process of posting multiple records. If the current field value is causing a problem, such as a key violation, when applying a cached update, datasets generate an *OnUpdateError* event. Similarly, *TProvider* objects generate an *OnUpdateError* event when problems occur posting records from a client, and client datasets generate an *OnReconcileError* event when informed of problems by the server. In the event handler, assign a new value to *NewValue* to correct the problem.

NewValue is the same as *Value*, except when errors are encountered while posting records. Setting *NewValue* in an *OnUpdateError* event handler, an *OnUpdateRecord* event handler, or an *OnReconcileError* event handler will cause *NewValue* to differ from *Value* until the records have finished being posted to the underlying database table.

See also *CurValue* property, *OldValue* property, *TClientDataSet.OnReconcileError* event, *TDataset.CachedUpdates* property, *TDataSet.OnUpdateError* event, *TDataSet.OnUpdateRecord* event, *TProvider.OnUpdateError* event

Offset

Offset is the number of bytes added to the end of the current record buffer in the dataset to store cached values for calculated fields or BLOB fields.

```
property Offset: word;
```

Description Read *Offset* to determine the extra space in the record buffer of a *TClientDataSet* object allocated for cached information on calculated fields or BLOB fields. *Offset* should only be used by component writers when developing new descendants of *TClientDataSet*.

See also *Calculated* property, *IsBlob* method, *TClientDataSet* object

OldValue

OldValue represents the original value of the field (as a *Variant*).

```
property OldValue: Variant;
```

Description Read the *OldValue* property to examine or retrieve the original value of the field that was obtained from the dataset before any new values were written to the record buffer. For example, this code replaces current pending changes with a field's original value:

```
NewValue := OldValue;
```

Once records are posted successfully, the old field value cannot be retrieved.

See also *CurValue* property, *NewValue* property

Origin

Origin is the name of the field in its original dataset.

property Origin: string;

Description Use the *Origin* property to distinguish the name of the field as it appears in its dataset from the name of the field on which it is based. For example, in a query that uses the following SQL statement, the value of *Origin* is CUSTOMER.CUSTNO.

```
SELECT CUSTNO AS ID FROM CUSTOMER
```

In this case, the *FieldName* property is ID.

Origin is especially useful when working with multi-table joins, as it distinguishes which table the field came from.

Note *Origin* is only filled in at design time by the Fields editor, and only when the field component is used by a *TQuery* object.

See also *FieldName* property

ReadOnly

ReadOnly determines whether the field can be modified.

property ReadOnly: Boolean;

Description Set *ReadOnly* to *True* to prevent a field from being modified. In a data grid, tabbing from field to field skips over *ReadOnly* fields.

To learn whether a field can be modified, use the *CanModify* property instead. *CanModify* reflects not only the value of the *ReadOnly* property, but also any restrictions imposed by the underlying dataset.

See also *CanModify* property

Required

Required specifies whether a nonblank value for a field is required.

property Required: Boolean;

Description Use *Required* to find out if a field requires a value or if the field can be blank.

If a field is created with the *Fields* editor, then this property is set based on the underlying table. Applications that set *Required* to *True* for fields that must have values (for example, a password or part number), but for which the underlying table does not require the field, must write an *OnValidate* event handler to enforce the property.

When the *Required* property reflects a property of the underlying database table, trying to post a null value will cause an exception to be raised. Applications that set the *Required* property to *True* when the underlying table does not require the field, should raise an exception on **nil** values in the *OnValidate* event handler in order to achieve the same result.

See also *OnValidate* event, *Validate* method

Size

Size indicates the size used in the definition of the physical database field for data types that support different sizes.

property Size: Word;

Description The interpretation of *Size* depends on the data type. *Size* is the size Borland DataBase Engine associates with the field. The meaning of *Size* for different data types is given in the following table:

DataType	Meaning of Size
ftBoolean	Size is not used; its value is always 0.
ftSmallInt	Size is not used; its value is always 0.
ftWord	Size is not used; its value is always 0.
ftAutoInc	Size is not used; its value is always 0.
ftDate	Size is not used; its value is always 0.
ftInteger	Size is not used; its value is always 0.
ftTime	Size is not used; its value is always 0.
ftCurrency	Size is not used; its value is always 0.
ftDateTime	Size is not used; its value is always 0.
ftFloat	Size is not used; its value is always 0.
ftBCD	Size is the number of digits after the decimal place.
ftString	Size is the maximum number of characters in the string.
ftVarBytes	Size is the maximum number of bytes of data, not counting the first two bytes which indicate the actual number of bytes of data that were used.
ftBytes	Size is the maximum number of bytes of data.
ftBlob	Size is the number of bytes from the BLOB that are stored in the record buffer.
ftDBaseOle	Size is the number of bytes from the DBase OLE BLOB that are stored in the record buffer.
ftFmtMemo	Size is the number of bytes from the memo that are stored in the record buffer.
ftGraphic	Size is the number of bytes from the image that are stored in the record buffer.
ftMemo	Size is the number of characters from the memo that are stored in the record buffer.
ftParadoxOle	Size is the number of bytes from the Paradox OLE BLOB that are stored in the record buffer.
ftTypedBinary	Size is the number of bytes from the BLOB that are stored in the record buffer.
ftUnknown	Size is not used; its value is always 0.
ftCursor	Size is not used; its value is always 0.

See also *DataSize* property, *DataType* property

Text

Text contains the string to display in a data-aware control when the field is in edit mode.

```
property Text: string;
```

Description Data-aware controls rely on the *Text* property to provide the editing format for each field. For example, by default the *Text* property of a currency field omits the thousands separators and currency symbol.

Text can differ from the *DisplayText* property if the field should have a different string representation when the value is being edited. To implement two different string representations of a field's value, use the *OnGetText* event handler. If an *OnGetText* event handler is assigned, *Text* is the value returned in the *Text* parameter of the *OnGetText* event handler when the *DisplayText* parameter is *False*. Otherwise, *Text* is the value of the *AsString* property.

See also *AsString* property, *DisplayText* property, *OnGetText* event, *Value* property

ValidChars

ValidChars is the set of characters that can make up the text representation of the value of the field.

```
type TFieldChars = set of Char;

property ValidChars: TFieldChars;
```

Description Set *ValidChars* to restrict the characters that can be validly entered as part of the text representation of the field's value. *ValidChars* is used by the *IsValidChar* method to determine whether a character typed into a data-aware edit control can make up part of the field's value. Unlike the *EditMask* property which specifies which characters can appear in specific positions, *ValidChars* limits the characters that can appear anywhere in the field's value.

TField initializes *ValidChars* to include all characters. Numeric descendants of *TField* change the default value to include only the characters that can make up a numeric value. Applications can change this value to restrict the character set for a field in other ways or to allow characters that are not permitted by default.

See also *EditMask* property, *IsValidChar* method

Value

Value contains the actual data in a field component.

```
property Value: Variant
```

Description Use *Value* to read data directly from and write data directly to a field component at runtime. For example, use the *Value* property to affect the contents of a field that is not *Visible*.

See also *Text* property, *DisplayText* property, *NewValue* property, *Visible* property

Visible

Visible determines whether the field appears in a data grid.

```
property Visible: Boolean;
```

Description To hide a field in a data grid, set *Visible* to *False*. If *Visible* is *True*, the field appears in the grid.

Note The *Visible* property has no effect if the grid has an explicit *Columns* value for the field. It also has no effect on controls other than *TDBGrid*, such as *TDBEdit*.

See also *TCustomDBGrid.Columns* property

Events

The following table lists the events of *TField*:

Event	Visibility	Documented in
OnChange	published	TField
OnGetText	published	TField
OnSetText	published	TField
OnValidate	published	TField

OnChange

The *OnChange* event occurs immediately after the data in the field is written to the record buffer.

```
type TFieldNotifyEvent = procedure(Sender: TField) of object;

property OnChange: TFieldNotifyEvent;
```

Description Write an *OnChange* event handler to respond to any changes in the field's data. *OnChange* allows a response once the data has been successfully written to the record buffer.

When writing the value of a field to the current record buffer, the following steps occur:

1. The *OnValidate* event handler is called to validate the data.
2. If the *OnValidate* event handler accepts the current *Value*, the data is written to the record buffer.

3 If writing the data does not raise an exception, the *OnChange* event handler is called to allow a response to the change.

See also *OnValidate* event, *TDataSet.AfterPost* event, *Value* property

OnGetText

The *OnGetText* event occurs whenever the *DisplayText* or *Text* properties of the field are referenced.

```
type TFieldGetTextEvent = procedure(Sender: TField; var Text: string; DisplayText: Boolean)
  of object;

property OnGetText: TFieldGetTextEvent;
```

Description Write an *OnGetText* handler to perform custom processing for the *DisplayText* and *Text* properties. Use *OnGetText* to format the *Value* of the field differently when it must be edited from the format used when simply displaying the value, or use *OnGetText* to display something other than the field's value when it is displayed.

The *Sender* parameter is the field component. The *Text* parameter receives the formatted text that is used by the *DisplayText* or *Text* property. The *DisplayText* parameter indicates whether the text will be used for display purposes only, or whether the string will be used for editing purposes.

If there is no *OnGetText* handler, the value of the *DisplayText* and *Text* properties is simply the value of the *AsString* property.

See also *AsString* property, *DisplayText* property, *OnSetText* event, *Text* property, *Value* property

OnSetText

The *OnSetText* occurs whenever the *Text* property is assigned a value.

```
type TFieldSetTextEvent = procedure(Sender: TField; const Text: string) of object;

property OnSetText: TFieldSetTextEvent;
```

Description Write an *OnSetText* event handler to specify processing that occurs whenever *Text* receives a new value. If there is an *OnGetText* event handler that formats the *Value* of the field in some special way for editing purposes, *OnSetText* should reverse the process when parsing the edited string.

If there is no *OnSetText* handler, the *Text* property will be set by the *AsString* property setting.

See also *AsString* property, *OnGetText* event, *Text* property, *Value* property

OnValidate

The *OnValidate* event occurs just before the data is written to the record buffer.

```
type TFieldNotifyEvent = procedure(Sender: TField) of object;
property OnValidate: TFieldNotifyEvent;
```

Description Write an *OnValidate* event handler to validate changes made to the data in the field, just before the data is written to the current record buffer. The *EditMask* property allows validation of the data on a character by character basis while it is being entered by the user. *OnValidate* allows an application to validate the data as a whole.

When the value of a field component is assigned programmatically, validation by the *EditMask* is bypassed, since there is no data-aware control to enforce adherence to the mask. *OnValidate* allows an application to validate such data before it is posted to the database table.

To reject the current value of the field from the *OnValidate* event handler, raise an exception.

When writing the value of a field to the current record buffer, the following steps occur:

1. The *OnValidate* event handler is called to validate the data.
2. If the *OnValidate* event handler does not raise an exception, the data is written to the current record buffer.
3. If writing the data does not raise an exception, the *OnChange* event handler is called to allow a response to the change.

See also *EditMask* property, *IsValidChar* method, *OnChange* event, *TDataSet.BeforePost* event, *Validate* method, *Value* property

Methods

The following table lists the methods of *TField*:

Method	Visibility	Documented in
Assign	public	TField
AssignValue	public	TField
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TField
Create	public	TField
DefaultHandler	public	TObject
Destroy	public	TField
DestroyComponents	public	TComponent

Method	Visibility	Documented in
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
FocusControl	public	TField
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetData	public	TField
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
IsBlob	public	TField
IsValidChar	public	TField
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RefreshLookupList	public	TField
RemoveComponent	public	TComponent
SafeCallException	public	TComponent
SetData	public	TField
SetFieldType	public	TField
Validate	public	TField

Assign

Assign assigns the *Value* property of one field to another or assigns another object to the field component.

```
procedure Assign(Source: TPersistent);
```

Description Use *Assign* to copy the *Value* property from one field to another. The fields must have compatible data types.

When *Source* is another field object, *Assign* calls the *AssignValue* method, using the *Source* field's *Value* property. Calling *Assign* with the *Source*

parameter set to **nil** assigns a blank value to the field component. Calling *Assign* with any other object type calls the inherited method, which allows assignment from objects that implement a *TField* object in their *AssignTo* method.

See also *AssignValue* method, *Clear* method, *DataType* property, *TPersistent.AssignTo* method, *Value* property

AssignValue

AssignValue sets the field to *Value* using the *AsInteger*, *AsBoolean*, *AsString* or *AsFloat* property, depending on the type of *Value*.

```
procedure AssignValue(const Value: TVarRec);
```

Description Use *AssignValue* to assign a value to a field using the *As...* conversion properties. The property used depends on the type of the *Value* parameter:

Value	As... property used
vtInteger	AsInteger
vtBoolean	AsBoolean
vtChar	AsString
vtExtended	AsFloat
vtString	AsString
vtPChar	AsString
vtAnsiString	AsString
vtCurrency	AsCurrency
vtVariant	AsVariant

If *Value* is not one of the types in the table, but is of type *TObject* or a *TObject* descendant, *AssignValue* uses the *Assign* method to assign the value to the field.

See also *AsBoolean* property, *AsCurrency* property, *AsFloat* property, *AsInteger* property, *Assign* method, *AsString* property

Clear

Clear sets the value of the field to NULL.

```
procedure Clear;
```

Description Call the *Clear* method to give the field a blank value.

See also *IsNull* property, *Required* property

Create

Create creates and initializes a field component.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to construct and initialize a new *TField* instance and inserts the newly-constructed control into its owner, as specified by the *AOwner* parameter. After calling the inherited constructor, *Create* initializes the *Visible* property to *True*.

Do not call *Create* directly for *TField*. *TField* is an abstract base class that should not be directly instantiated. *The Create method for* descendant classes is usually called indirectly by using the Fields editor at design time to create a persistent field component. Dataset objects also call *Create* automatically to create dynamic field components.

See also *Visible* property

Destroy

Destroy destroys an instance of *TField*.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. *Destroy* is called automatically by the dataset that owns the field component, and applications never need to call it explicitly. If an application creates a descendant of *TField* without an owner, it should call *Free* instead of *Destroy*. *Free* verifies that the field object has not already been freed, and only then calls *Destroy*.

Destroy closes the dataset, deletes the instance of *TField from the dataset*, and calls the inherited destructor.

See also *DataSet* property, *Free* method, *TComponent.Destroy* method

FocusControl

FocusControl enables a field object to force a form's focus to the first data-aware component associated with the field object.

function FocusControl;

Description Call *FocusControl* to switch the focus of the form to a data-aware control associated with the field. Because it's possible that the field's value can change even if its associated data-aware control doesn't have the focus, calling *FocusControl* can aid in drawing attention to the new value when it fails to be validated.

See also *OnValidate* event, *Validate* method

GetData

GetData returns the unformatted data from the field.

function GetData(Buffer: Pointer): Boolean;

Description Call *GetData* to return the unformatted, untranslated, and uninterpreted data in a field to a buffer.

Unlike the *DisplayText*, *Text*, or *AsXXX* properties, *GetData* performs no translation or interpretation of the data. *Buffer* must have sufficient space allocated for the data. Use the *DataSize* property to determine the space required. If the data is NULL, *GetData* returns *False* and no data is transferred to *Buffer*. Otherwise, *GetData* returns *True*.

Note *GetData* cannot be used to read data from BLOB fields.

See also *DataSize* property, *DisplayText* property, *IsBlob* method, *SetData* method, *Text* property

IsBlob

IsBlob indicates whether the field represents the data in a binary large object (BLOB) field.

```
class function IsBlob: Boolean; virtual;
```

Description Call *IsBlob* to determine whether the field represents BLOB data. For example, before using the *GetData* or *SetData* method, call *IsBlob* to determine whether the method can be used. The *IsBlob* method for *TField* always returns *False*. Descendants of *TField* that implement blob fields override this method to indicate that they represent BLOB data.

See also *GetData* method, *SetData* method

IsValidChar

IsValidChar indicates if a particular character is valid for the field.

```
function IsValidChar(InputChar: Char): Boolean; virtual;
```

Description Call *IsValidChar* to determine if a character specified as the value of the *InputChar* can appear in the text representation of the value of the field. Many data-aware controls use *IsValidChar* to check characters entered by the user to determine if they are valid. *IsValidChar* returns *True* if *InputChar* can appear in the field. *IsValidChar* returns *False* if *InputChar* is not a valid character for the field type.

The valid characters for a field object are given by the *ValidChars* property. The default valid characters for each *DataType* are given in the following table.:

DataType	Valid characters
ftBoolean	All characters.
ftSmallInt	Digits 0 to 9, plus sign (+), and minus sign (–).
ftWord	Digits 0 to 9, plus sign (+), and minus sign (–)
ftAutoInc	Digits 0 to 9, plus sign (+), and minus sign (–)

DataType	Valid characters
ftDate	All characters.
ftInteger	Digits 0 to 9, plus sign (+), and minus sign (–)
ftTime	All characters
ftCurrency	Digits 0 to 9, plus sign (+), minus sign (–), the letter E (E or e) and the decimal separator from the regional settings in the Windows control panel.
ftDateTime	All characters.
ftFloat	Digits 0 to 9, plus sign (+), minus sign (–), the letter E (E or e) and the decimal separator from the regional settings in the Windows control panel
ftBCD	Digits 0 to 9, plus sign (+), minus sign (–), and the decimal separator from the regional settings in the Windows control panel.
ftString	All characters.
ftVarBytes	All characters.
ftBytes	All characters.
ftBlob	All characters.
ftDBaseOle	All characters.
ftFmtMemo	All characters.
ftGraphic	All characters.
ftMemo	All characters.
ftParadoxOle	All characters.
ftTypedBinary	All characters.
ftUnknown	All characters.
ftCursor	Does not apply to *TField*.

Note For some *DataTypes*, such as *ftGraphic*, *IsValidChar* returns *True* for all characters although not every character is valid for the field. *IsValidChar* provides only a rough validation of input characters. More refined character-by-character validation can be provided by an edit mask. Also, the value for a field is validated as a whole when it is written to the database record.

See also *DataType* property, *EditMask* property, *OnValidate* event, *ValidChars* property

RefreshLookupList

RefreshLookupList loads *LookupList* with all the values for *LookupResultField* that match distinct values of *LookupKeyFields*.

```
procedure RefreshLookupList;
```

Description Call *RefreshLookupList* to load the lookup cache when setting the *LookupCache* property at runtime or when the values of the *LookupResultField* or the *LookupKeyFields* in the *LookupDataSet* may have changed.

See also *LookupCache* property, *LookupDataSet* property, *LookupKeyFields* property, *LookupList* property, *LookupResultField* property

SetData

SetData assigns unformatted data to the field.

```
procedure SetData(Buffer: Pointer);
```

Description Use *SetData* to assign data to a field in the format that is used by the underlying database table. Ultimately, any method of setting the data for the field component calls *SetData*, once the data has been parsed into the physical format expected by the database table.

SetData calls the *OnValidate* event handler before writing the data to the current record buffer. If the data is successfully written, *SetData* calls the *OnChange* event handler.

Note *SetData* should not be used by descendants of *TField* that implement BLOB field types.

See also *GetData* method, *IsBlob* method, *OnChange* event, *OnValidate* event

SetFieldType

SetFieldType provides the interface for a method that can set the *DataType* for the field.

```
type TFieldType = (ftUnknown, ftString, ftSmallint, ftInteger, ftWord, ftBoolean, ftFloat,
  ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftBytes, ftVarBytes, ftAutoInc, ftBlob, ftMemo,
  ftGraphic, ftFmtMemo, ftParadoxOle, ftDBaseOle, ftTypedBinary, ftCursor);

procedure SetFieldType(Value: TFieldType);
```

Description *SetFieldType* does nothing other than provide an interface for derived classes to override *to implement a method that sets the DataType for the field component*. *SetFieldType* is useful for derived classes that can support more than one underlying data type. For example, a *TBlobField* may support *ftBlob*, *ftFmtMemo*, *ftParadoxOle*, *ftDBaseOle*, or *ftTypedBinary*.

See also *DataType* property, *TBlobField* object

Validate

Validate calls the *OnValidate* event handler if it is assigned.

```
procedure Validate(Buffer: Pointer);
```

Description Call *Validate* to validate the unformatted data pointed to by the *Buffer* parameter.*Validate* is called by the dataset when writing new values to the current record.

Before calling the *OnValidate* event handler, *Validate* sets up the unformatted data in *Buffer* so that the properties that represent the field's value will return formatted versions of this value. The value of *Buffer* does not affect the value of the field outside the *OnValidate* event handler.

See also *OnValidate* event, *Value* property

TFileListBox

Unit: FileCtrl | Palette: Win3.1

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomListBox • TFileListBox

Description

TFileListBox is a specialized list box that lists all the files in a specified directory.

Add *TFileListBox* to a form to allow users to select a file. Use *TFileListBox* along with *TDriveComboBox*, *TFilterComboBox*, and *TDirectoryListBox* to add full file selection capabilities to a form.

To add a standard Windows 95 File Open or Save dialog to an application, use *TOpenDialog* or *TSaveDialog* instead.

See also *TDirectoryListBox* object, *TDriveComboBox* object, *TFilterComboBox* object, *TOpenDialog* object, *TSaveDialog* object

Properties

The following table lists the properties of *TFileListBox*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Canvas	public, read-only		TCustomListBox
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl

Property	Visibility	Default	Documented in
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
Directory	public		TFileListBox
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Drive	public		TFileListBox
Enabled	published	True	TControl
ExtendedSelect	published	True	TCustomListBox
FileEdit	published		TFileListBox
FileName	public		TFileListBox
FileType	published	[ftNormal]	TFileListBox
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
ImeMode	published	imDontCare	TWinControl
ImeName	published		TWinControl
IntegralHeight	published	False	TCustomListBox
ItemHeight	published		TCustomListBox
ItemIndex	public		TCustomListBox
Items	public		TCustomListBox
Left	published		TControl
Mask	published		TFileListBox
MultiSelect	published	False	TCustomListBox
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
SelCount	public, read-only		TCustomListBox
Selected	public		TCustomListBox
ShowGlyphs	published	False	TFileListBox
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl

Property	Visibility	Default	Documented in
Tag	published	0	TComponent
Top	published		TControl
TopIndex	public		TCustomListBox
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Directory

Directory is the directory that contains all the files listed in the file list box.

property Directory: **string**;

Description Set *Directory* to update the file list box so that it displays the files in a new directory. If the form includes a *TDirectoryListBox* object, set the *FileList* property of the directory list box to have the *Directory* property updated automatically when the user selects a new directory.

See also *Drive* property, *TDirectoryListBox.Directory* property, *TDirectoryListBox.FileList* property

Drive

Drive is the drive which contains the files listed in the list box.

property Drive: Char;

Description Set the value of *Drive* to the drive that contains the files that the list box should display. When the value of *Drive* changes, the *Directory* value also changes to the current directory on the specified drive.

If the form includes a *TDirectoryListBox* object, set the *FileList* property of the directory list box to have the *Drive* property updated automatically when the *Drive* property of the directory list box changes.

See also *Directory* property, *TDirectoryListBox.Drive* property, *TDirectoryListBox.FileList* property

FileEdit

FileEdit links the directory list box to an edit control that displays the currently selected file.

property FileEdit: TEdit;

Description Set *FileEdit* to an edit control that displays the selected file from the file list box, as is commonly done in Open and Save dialog boxes. If no file is selected in the file list box, the text of the edit control is the current value of the file list box's *Mask* property.

Note If the user changes the text of the edit control, the results are not automatically reflected in the file list box. Use the *ApplyFilePath* method to update the file list box when the text of the edit control changes.

See also *ApplyFilePath* method, *FileName* property, *Mask* property, *TEdit* object

FileName

FileName is the name of the selected file in the list box, including the path name.

```
property FileName: string;
```

Description Set *FileName* to initialize the file list box to a particular file. Read *FileName* to obtain the name of the file the user selected.

If the *FileEdit* property is set, the linked edit control displays the value of *FileName* whenever the *FileName* property changes.

See also *FileEdit* property, *FileType* property

FileType

FileType determines which files are displayed in the file list box based on the attributes of the files.

```
type
  TFileAttr = (ftReadOnly, ftHidden, ftSystem, ftVolumeID, ftDirectory, ftArchive, ftNormal);
  TFileType = set of TFileAttr;

property FileType: TFileType;
```

Description Set *FileType* to the type of files the list box should display. *FileType* can specify more than one type of file. For example, if the value of *FileType* is a set containing the values *ftReadOnly* and *ftHidden*, files that have either the read-only or the hidden attributes are displayed in the list box. These are the values that can occur in the *FileType* property:

Value	Meaning
ftReadOnly	The list box displays files with the read-only attribute.
ftHidden	The list box displays files with the hidden attribute.
ftSystem	The list box displays system files.
ftVolumeID	The list box displays the volume name.
ftDirectory	The list box displays directories.
ftArchive	The list box displays files with archive attribute.
ftNormal	The list box displays files with no special attributes.

To set the *FileType* property using the Object Inspector, click the *FileType* property to see the attribute values. Then set each value to *True* or *False* to specify whether it is included in the *FileType* set.

Note The files that are listed can also be limited by the form of their names. To restrict the files to particular names, use the *Mask* property or include a *TFilterComboBox* on the form.

See also *Mask* property, *TFilterComboBox* object

Mask

Mask limits which files are displayed in the file list box.

```
property Mask: string;
```

Description Set *Mask* to a regular expression to limit the list box to files that match the mask. The value of the mask is a file name that may include wildcards. The asterisk (*) is a wildcard which matches any number of arbitrary characters. The question mark (?) is a wildcard that matches a single arbitrary character. The file mask *.* displays all files, which is the default value.

To specify multiple file masks, separate the file mask specifications with semicolons. For example, *.PAS; *.EXE.

If the form includes a *TFilterComboBox* object, set the *FileList* property of the filter combo box to have the *Mask* property updated automatically when the *Mask* property of the filter combo box changes.

See also *FileEdit* property, *FileName* property, *TFilterComboBox.FileList* property, *TFilterComboBox.Mask* property

ShowGlyphs

ShowGlyphs determines whether glyphs (bitmaps) appear next to the file names listed in the file list box.

```
property ShowGlyphs: Boolean;
```

Description Set *ShowGlyphs* to *True* to display icons next to the files in the list that indicate the file type. Set *ShowGlyphs* to *False* to list only the file names.

Events

The following table lists the events of *TFileListBox*:

Event	Visibility	Documented in
OnChange	published	TFileListBox
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl

Event	Visibility	Documented in
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

OnChange

OnChange occurs immediately after the user changes the selected file in the list.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnChange: TNotifyEvent;
```

Description Write an *OnChange* event handler to take specific action when the *FileName* property changes. If the *FileEdit* property has been set, the edit control is updated before the *OnChange* event.

See also *FileEdit* property, *FileName* property

Methods

The following table lists the methods of *TFileListBox*:

Method	Visibility	Documented in
ApplyFilePath	public	TFileListBox
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TCustomListBox
ClientToScreen	public	TControl
ContainsControl	public	TWinControl

Method	Visibility	Documented in
ControlAtPos	public	TWinControl
Create	public	TFileListBox
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TFileListBox
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
ItemAtPos	public	TCustomListBox
ItemRect	public	TCustomListBox
MethodAddress	public	TObject
MethodName	public	TObject

Method	Visibility	Documented in
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TFileListBox
UpdateControlState	public	TWinControl

ApplyFilePath

ApplyFilePath updates the file list box to represent a fully qualified file name.

```
procedure ApplyFilePath(const EditText: string);
```

Description Call *ApplyFilePath* to change the *Directory*, *Drive*, *FileName*, and *Mask* properties so that they match the values implied by the *EditText* parameter. For example, if an edit control is linked to the file list box using the *FileEdit* property, call *ApplyFilePath* from the *OnChange* event handler of the edit control to update the file list box whenever the user types a value in the edit control.

ApplyFilePath updates the *Mask* property only if the *EditText* parameter includes wildcards. If the form contains a *TDirectoryListBox* object that specifies this file list box object in its *FileList* property, the *Directory* and *Drive* properties of the directory list box object are updated as well. If the form also includes a *TDriveComboBox* object that specifies the directory list box in its *DirList* property, the *Drive* property of the drive combo box will also be updated. However, the *Mask* property of any *TFilterComboBox* will not be updated.

See also *Directory* property, *Drive* property, *FileEdit* property, *FileName* property, *Mask* property, *TDirectoryListBox* object, *TDriveComboBox* object, *TFilterComboBox* object

Create

Create creates an instance of *TFileListBox*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to create a *TFileListBox* object at runtime. File list boxes placed on a form at design time are created automatically.

After allocating the memory for the list box, *Create*

- Initializes the *Mask* property to *.*.
- Initializes the *FileType* property to [*ftNormal*].
- Initializes the *Drive* and *Directory* properties to the current directory and fills the list box with all the files in the directory.
- Initializes the *MultiSelect* property to *False*.
- Initializes the *Sorted* property to *True*.
- Initializes the *Width* property to 145.

See also *Directory* property, *Drive* property, *FileType* property, *Mask* property, *MultiSelect* property, *Sorted* property, *TCustomListBox.Create* method, *Width* property

Destroy

Destroy destroys an instance of *TFileListBox*.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the file list box is not already freed, and only then calls *Destroy*.

See also *Free* method

Update

Update refreshes the directory list for the file list box.

procedure Update;

Description Call *Update* whenever the files listed by the file list box may be out of date. For example, if an application creates, renames, or deletes files, calling *Update* will ensure that those changes are reflected in the file list box.

The list box is updated automatically when the *Drive*, *Directory*, *FileType*, or *Mask* property is changed.

See also *Directory* property, *Drive* property, *FileType* property, *Mask* property

TFilterComboBox

Unit: FileCtrl

Palette: Win3.1

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomComboBox • TFilterComboBox

Description

TFilterComboBox is a specialized combo box that presents the user with a choice of file filters.

Add a filter combo box to a form that includes a file selection control to provide a set of predefined file filters. Use *TFilterComboBox* along with *TDriveComboBox*, *TDirectoryListBox*, and *TFileListBox* to add full file selection capabilities to a form.

To add a standard Windows 95 File Open or Save dialog to an application, use *TOpenDialog* or *TSaveDialog* instead.

See also

TDirectoryListBox object, *TDriveComboBox* object, *TFileListBox* object

Properties

The following table lists the properties of *TFilterComboBox*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Canvas	public, read-only		TCustomComboBox
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent

Property	Visibility	Default	Documented in
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
DroppedDown	public		TCustomComboBox
Enabled	published	True	TControl
FileList	published		TFilterComboBox
Filter	published		TFilterComboBox
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
ImeMode	published	imDontCare	TWinControl
ImeName	published		TWinControl
ItemIndex	public		TCustomComboBox
Items	public		TCustomComboBox
Left	published		TControl
Mask	public, read-only		TFilterComboBox
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
SelLength	public		TCustomComboBox
SelStart	public		TCustomComboBox
SelText	public		TCustomComboBox
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent
Text	public		TControl
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

FileList

FileList links the filter combo box to a file list box control.

property FileList: TFileListBox;

Description If the form includes a *TFileListBox* object to display a list of files to choose from, set *FileList* to allow the filter combo box to limit which files are displayed in the list box. Once the two controls are connected, whenever a new filter is selected using a filter combo box, the file list box displays the files that match the selected filter.

See also *Mask* property, *TFileListBox* object

Filter

Filter contains all the file masks displayed in the filter combo box.

property Filter: **string**;

Description Set the *Filter* property to specify the names that appear in the filter combo box, along with the masks associated with each name. Each mask is a file name that may include wildcards. The asterisk (*) is a wildcard that matches any number of arbitrary characters. The question mark (?) is a wildcard that matches a single arbitrary character. When the application runs, the file filter the user selects in the filter combo box becomes the value of the *Mask* property.

To create the value of Filter for a single file type, follow these steps:

1. Type some meaningful text to indicate the type of file.
2. Type a | character (vertical bar).
3. Type the file filter.

Don't put in any spaces around the | character in the string.

Here's an example:

```
FilterComboBox1.Filter := 'Text files|*.TXT';
```

If Filter is set to this string, the string "Text files" appears in the filter combo box. When the user selects "Text files", the mask is set to "*.TXT".

To specify multiple file filters, extend the value of *Filter*, separating each file name/mask combination with a | character. *Filter can* specify an any number of file filters, as long as the string is less than 255 characters.

Here's an example of two file filters that could be specified as the value of the *Filter* property:

```
'Text files (*.TXT)|*.TXT|Pascal files (*.PAS)|*.PAS'
```

The previous example includes the file filters in parentheses in the text parts. This isn't required, but it's a common convention that helps users understand what to expect when they select a file filter.

The mask portion can include multiple file specifiers if they are separated with semicolons:

```
'All files|*.TXT;*.PAS;*.WB1'
```

See also *Mask* property

Mask

Mask is the mask associated with the string selected in the filter combo box.

property Mask: **string**;

Description Read *Mask* to obtain the mask that was selected by the user.

When the user uses the filter combo box to select a new filter, the mask associated with the selected file type becomes the value of the *Mask* property. If a file list box is linked to the filter combo box with the *FileList* property, the file list will update when the value of *Mask* changes.

See also *FileList* property, *Filter* property, *TFileListBox.Mask* property

Events

The following table lists the events of *TFilterComboBox*:

Event	Visibility	Documented in
OnChange	published	TCustomComboBox
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnDropDown	published	TCustomComboBox
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnStartDrag	published	TControl

OnChange

OnChange occurs immediately after the user selects a new filter using the filter combo box.

type TNotifyEvent = **procedure**(Sender: TObject) **of** object;

property OnChange: TNotifyEvent;

Description Write an *OnChange* event handler to take specific action when the *Mask* property changes. If the *FileList* property has been set, the file list box is updated before the *OnChange* event.

For example, if an application needed to synchronize two file list boxes with a single filter combo box, the file list box that was not updated automatically because of the *FileList* property could be synchronized with the filter combo box by writing the following *OnChange* event handler:

```
procedure TForm1.FilterComboBox1Change(Sender: TObject);
begin
  FileListBox2.Mask := TFilterComboBox(Sender).Mask;
end;
```

See also *FileList* property, *Mask* property, *TFileListBox* object

Methods

The following table lists the methods of *TFilterComboBox:*

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TCustomComboBox
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TFilterComboBox
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TFilterComboBox
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl

Method	Visibility	Documented in
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SelectAll	public	TCustomComboBox

Method	Visibility	Documented in
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

Create

Create creates and initializes an instance of *TFilterComboBox*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to create a *TFilterComboBox* object at runtime. Filter combo boxes placed on a form at design time are created automatically.

After allocating the memory for the combo box, *Create* initializes the *Mask* property to *.* and initializes the *Style* property to *csDropDownList*.

See also *Mask* property, *Style* property, *TCustomComboBox.Create* method

Destroy

Destroy destroys an instance of *TFilterComboBox*.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the filter combo box is not already freed, and only then calls *Destroy*.

See also *Free* method

TFindDialog

Unit: Dialogs | Palette: Dialogs

TObject • TPersistent • TComponent • TCommonDialog • TFindDialog

TFindDialog displays a Find dialog that allows users to search for text in a file.

Description The *TFindDialog* component displays a modeless Windows dialog box that prompts the user for a search string. The dialog does not appear at runtime until it is activated by a call to the *Execute* method.

See also *TReplaceDialog* object

Properties

The following table lists the properties of *TFindDialog*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Ctl3D	published	True	TCommonDialog
DesignInfo	public		TComponent
FindText	published		TFindDialog
Handle	public, read-only		TCommonDialog
HelpContext	published	0	TCommonDialog
Left	public		TFindDialog
Name	published		TComponent
Options	published	[frDown]	TFindDialog
Owner	public, read-only		TComponent
Position	public		TFindDialog
Tag	published	0	TComponent
Top	public		TFindDialog
VCLComObject	public		TComponent

FindText

Contains the search string.

property FindText: **string**;

Description *FindText* contains the text string that the user wants to search for. When the user clicks the Find Next button, the text in the dialog's edit box is automatically assigned to *FindText*. To make a default text string appear in the edit box when the dialog opens, assign a value to *FindText* in the Object Inspector or in program code.

Left

Determines the X coordinate (horizontal position) of the Find dialog's upper left corner.

property Left: Integer;

Description *Left* holds the same value as the first integer in *Position*. This property is maintained for compatibility with older applications.

See also *Position* property, *Top* property

Options

Determines the appearance and behavior of the Find dialog.

```
type
  TFindOption = (frDown, frFindNext, frHideMatchCase, frHideWholeWord, frHideUpDown,
    frMatchCase, frDisableMatchCase, frDisableUpDown, frDisableWholeWord, frReplace,
    frReplaceAll, frWholeWord, frShowHelp);
  TFindOptions = set of TFindOption;

property Options: TFindOptions;
```

Description The possible values of *Options* are

Value	Meaning
frDisableMatchCase	Disables (grays) the Match Case check box.
frDisableUpDown	Disables (grays) the Up and Down buttons, used to determine the direction of the search.
frDisableWholeWord	Disables (grays) the Match Whole Word check box.
frDown	Selects the Down button by default when the dialog opens. If the *frDown* flags is off, Up is selected when the dialog opens. (By default, *frDown* is on.)
frFindNext	This flag is turned on when the user clicks the Find Next button and turned off when the dialog closes.
frHideMatchCase	Removes the Match Case check box from the dialog.
frHideWholeWord	Removes the Match Whole Word check box from the dialog.
frHideUpDown	Removes the Up and Down buttons from the dialog.
frMatchCase	This flag is turned on (off) when the user selects (deselects) the Match Case check box. To select the check box by default when the dialog opens, set *frMatchCase* at design time.
frReplace	Applies to *TReplaceDialog* only. This flag is set by the system to indicate that the application should replace the current occurrence (and only the current occurrence) of the *FindText* string with the *ReplaceText* string.
frReplaceAll	Applies to *TReplaceDialog* only. This flag is set by the system to indicate that the application should replace all occurrences of the *FindText* string with the *ReplaceText* string.
frShowHelp	Displays a Help button in the dialog.
frWholeWord	This flag is turned on (off) when the user selects (deselects) the Match Whole Word check box. To select the check box by default when the dialog opens, set *frWholeWord* at design time.

By default, all options except *frDown* are off.

See also *FindText* property, *TReplaceDialog* object, *TReplaceDialog.OnReplace* event, *TReplaceDialog.ReplaceText* property

Position

Determines the position of the Find dialog on the screen.

property Position: TPoint;

Description *Position* gives the pixel coordinates of the dialog's upper left corner.

See also *Left* property, *Top* property, *TPoint* type

Top

Determines the Y coordinate (vertical position) of the Find dialog's upper left corner.

property Top: Integer;

Description *Top* holds the same value as the second integer in *Position*. This property is maintained for compatibility with older applications.

See also *Left* property, *Position* property

Events

The following table lists the properties of *TFindDialog*:

Event	Visibility	Documented in
OnClose	published	TCommonDialog
OnFind	published	TFindDialog
OnShow	published	TCommonDialog

OnFind

OnFind occurs when the user clicks the Find Next button.

property OnFind: TNotifyEvent;

Description When the user clicks Find Next, an *OnFind* event occurs and the *frFindNext* flag is set in *Options*. Write an *OnFind* event handler that searches for the text specified in *FindText*, using the *Options* flags to determine how the search is conducted.

For example, the following event handler searches a *TMemo* object called *Memo1* and highlights the located text using *TMemo*'s *SelStart* and *SelLength* properties.

```
procedure TForm1.FindDialog1Find(Sender: TObject);
var
  SelPos: Integer;
begin
  with TFindDialog(Sender) do
  begin
    { Perform a global case-sensitive search for FindText in Memo1 }
    SelPos := Pos(FindText, Memo1.Lines.Text);
    if SelPos > 0 then
    begin
      Memo1.SelStart := SelPos - 1;
      Memo1.SelLength := Length(FindText);
    end
    else MessageDlg(Concat('Could not find "', FindText, '" in Memo1.'), mtError,
      [mbOk], 0);
  end;
end;
```

See also *FindText* property, *Options* property, *TMemo* object, *TNotifyEvent* type

Methods

The following table lists the properties of *TFindDialog*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
CloseDialog	public	TFindDialog
Create	public	TFindDialog
DefaultHandler	public	TObject
Destroy	public	TFindDialog
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
Execute	public	TFindDialog
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject

Method	Visibility	Documented in
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

CloseDialog

Closes the Find dialog box.

```
procedure CloseDialog;
```

Description *CloseDialog* closes the dialog box and frees its window handle, but leaves most properties of the *TFindDialog* (including *FindText*) unchanged.

Create

Creates and initializes a *TFindDialog* instance.

```
constructor Create(AOwner: TComponent); override;
```

Description The *Create* method generates a *TFindDialog* instance, but the new dialog does not appear on the form at runtime until the *Execute* method is called.

See also *Execute* method

Destroy

Destroys the *TFindDialog* and frees its memory.

```
destructor Destroy; override;
```

Description Do not call *Destroy* directly in an application. Call *Free* instead. *Free* verifies that the Find dialog is not already destroyed before calling *Destroy*.

See also *Free* method

Execute

Displays the Find dialog.

```
function Execute: Boolean; override;
```

Description *Execute* opens the Find dialog, returning *True* when a window handle is successfully allocated. If the dialog is already open when *Execute* is called, *Execute* brings the dialog to the top, activates its parent form, and returns *True*.

TFloatField

Unit: DB

TObject • TPersistent • TComponent • TField • TNumericField • TFloatField

A *TFloatField* object represents a field that contains floating-point values in a dataset.

Description Floating-point fields can hold values in the range from (positive or negative) $5.0 * 10^{-324}$ to $1.7 * 10^{308}$ with an accuracy of 15 digits.

TFloatField introduces new properties to

- Convert between floating-point values and other data types.
- To limit the range or precision of valid values that the field will permit.
- To format the value for display purposes.

As a descendant of *TNumericField*, *TFloatField* includes many properties, methods, and events that are useful for managing the value and properties of a numeric field in a database.

TFloatField is the direct ancestor of the *TCurrencyField* component.

See also *TCurrencyField* object, *TDataSet* object

Properties

The following table lists the properties of *TFloatField*:

Property	Visibility	Default	Documented in
Alignment	published	taRightJustify	TField
AsBoolean	public		TField
AsCurrency	public		TField
AsDateTime	public		TField
AsFloat	public		TFloatField
AsInteger	public		TFloatField

Property	Visibility	Default	Documented in
AsString	public		TFloatField
AsVariant	public		TFloatField
AttributeSet	public		TField
Calculated	public	False	TField
CanModify	public, read-only		TField
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ConstraintErrorMessage	published		TField
Currency	published	False	TFloatField
CurValue	public, read-only		TField
CustomConstraint	published		TField
DataSet	public		TField
DataSize	public, read-only		TFloatField
DataType	public, read-only		TField
DefaultExpression	published		TField
DesignInfo	public		TComponent
DisplayFormat	published		TNumericField
DisplayLabel	published		TField
DisplayName	public, read-only		TField
DisplayText	public, read-only		TField
DisplayWidth	published		TField
EditFormat	published		TNumericField
EditMask	public		TField
EditMaskPtr	public, read-only		TField
FieldKind	published	fkData	TField
FieldName	published		TField
FieldNo	public, read-only		TField
ImportedConstraint	published		TField
Index	published		TField
IsIndexField	public, read-only		TField
IsNull	public, read-only		TField
KeyFields	published		TField
Lookup	public		TField
LookupCache	published	False	TField
LookupDataSet	published		TField
LookupKeyFields	published		TField
LookupList	public, read-only		TField
LookupResultField	published		TField

Property	Visibility	Default	Documented in
MaxValue	published		TFloatField
MinValue	published		TFloatField
Name	published		TComponent
NewValue	public		TField
Offset	public, read-only		TField
OldValue	public, read-only		TField
Origin	published		TField
Owner	public, read-only		TComponent
Precision	published	15	TFloatField
ReadOnly	published	False	TField
Required	published	False	TField
Size	public		TField
Tag	published	0	TComponent
Text	public		TField
ValidChars	public		TField
Value	public		TFloatField
VCLComObject	public		TComponent
Visible	published	True	TField

AsFloat

AsFloat gets or sets the value of the floating-point field.

property AsFloat: Double;

Description Floating-point values need no conversion, because the native storage format of a floating-point field is a Double. *AsFloat* provides a uniform interface that allows applications to get or set floating-point values with generic field components when the internal storage format of the data is not known.

If the value assigned to the field does not fall within the range specified by the *MinValue* and *MaxValue* properties, an *EDatabaseError* exception is raised.

See also *EDatabaseError* object, *MaxValue* property, *MinValue* property, *Value* property

AsInteger

AsInteger converts the value of the floating-point field to or from an integer value.

property AsInteger: Longint;

Description Use *AsInteger* to

- Assign an integer value to the floating-point field.
- Assign the floating-point field value to an integer.

AsInteger converts floating point values to integers by truncating the decimal portion. If a value assigned to the field is not within the range established by the *MinValue* and the *MaxValue* properties, an *EDatabaseError* exception is raised.

See also *EDatabaseError* object, *MaxValue* property, *MinValue* property

AsString

AsString converts the value of a floating-point field to or from a string value.

property AsString: **string**;

Description Use *AsString* to

- Assign the floating-point field value to string value.
- Assign a string value to the floating-point field.

For example, this code assigns a string value to floating-point field value:

```
MyFloatField.AsString := '13.5';
```

To assign floating-point field value to a string, use code similar to this:

```
NewString := MyFloatField.AsString;
```

If the string assigned to the field does not represent a valid floating-point value within the range specified by the *MinValue* and *MaxValue* properties, an *EDatabaseError* exception is raised.

See also *EDatabaseError* object, *MaxValue* property, *MinValue* property

AsVariant

AsVariant converts the value of a floating-point field to or from a variant value.

property AsVariant: Variant;

Description Use *AsVariant* to

- Assign the floating-point field value to a variant variable.
- Assign a variant value to the floating-point field.

Using variants is useful when the actual field type and the format of the data are unknown.

If the value assigned to the field does not fall within the range specified with *MinValue* and *MaxValue* properties, an *EDatabaseError* exception is raised.

See also *AsFloat* property, *AsInteger* property, *AsString* property, *EDatabaseError* object, *MaxValue* property, *MinValue* property

Currency

Currency determines whether the value in the field should be formatted as a currency value.

property Currency: Boolean;

Description Use the *Currency* property to specify the formatting of the field for viewing and editing in a data-aware control.

The field is formatted for display using the *DisplayFormat* property if it is assigned. If *DisplayFormat* is not assigned, *Currency* determines how the field is formatted for display. If *Currency* is *True*, the value is formatted for display using the *FloatToText* function with the *ffCurrency* formatting code. If *Currency* is *False*, the value is formatted with the *ffGeneral* format.

The field is formatted for editing purposes using the *EditFormat* (or *DisplayFormat*) property if it is assigned. If neither *EditFormat* nor *DisplayFormat* is assigned, *Currency* determines how the field is formatted for editing. If *Currency* is *True*, the value is formatted for editing using the *FloatToText* function with the *ffFixed* formatting code. If *Currency* is *False*, the value is formatted with the *ffGeneral* format.

See also *DisplayFormat* property, *DisplayText property*, *EditFormat* property, *FloatToText* function, *FloatToTextFmt* function, *Text property*

DataSize

DataSize is the amount of memory that is needed to store the value of a float field in its native database format.

property DataSize: Word;

Description Check *DataSize* to determine the number of bytes required to store a float field component's value. Use *DataSize* to determine the size needed for a buffer when working with the *GetData* and *SetData* methods.

For float fields, *DataSize* is always 8. Database tables store float fields as a Double, which is the same as the type used by *TFloatField*.

See also *AsFloat* property, *GetData* method, *SetData* method, *Value* property

MaxValue

MaxValue limits the maximum value in the floating-point field.

property MaxValue: Double;

Description Use *MaxValue* to get or set the maximum value that can be entered into the field. If a value greater than *MaxValue* is entered, an *EDatabaseError* exception is raised.

See also *EDatabaseError object*, *MinValue* property

MinValue

MinValue limits the minimum value in the floating-point field.

property MinValue: Double;

Description Use *MinValue* to get or set the minimum value that can be entered into the field. If a value smaller than *MinValue* is entered, an *EDatabaseError* exception is raised.

See also *EDatabaseError object*, *MaxValue* property

Precision

Precision determines the precision used in formatting the value in a floating-point field.

property Precision: Integer;

Description Use *Precision* set the number of digits beyond the decimal point the value should be formatted to before rounding begins. *Precision* affects the format of the floating-point value when it is displayed as a text string, not the actual floating-point value that is stored in the field.

The default value of *Precision* is 15 decimal places.

Note *Precision* must be greater than or equal to 2. Trying to set *Precision* to a value less than 2 changes the value of *Precision* to 2.

See also *AsString* property, *DisplayFormat* property, *DisplayText* property, *EditFormat* property, *Text* property

Value

Value is the actual data in the floating-point field.

property Value: Double;

Description Use *Value* to read data directly from and write data directly to a floating-point component at runtime. Using the *Value* property is the same as using the *AsFloat* property. Use *Value* when you know the field component is a floating-point field. Use *AsFloat* when working with a generic field component.

See also *AsFloat* property

Events

The following table lists the events of *TFloatField*:

Event	Visibility	Documented in
OnChange	published	TField
OnGetText	published	TField
OnSetText	published	TField
OnValidate	published	TField

Methods

The following table lists the methods of *TFloatField*:

Method	Visibility	Documented in
Assign	public	TField
AssignValue	public	TField
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TField
Create	public	TFloatField
DefaultHandler	public	TObject
Destroy	public	TField
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
FocusControl	public	TField
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetData	public	TField
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent

Method	Visibility	Documented in
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
IsBlob	public	TField
IsValidChar	public	TField
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RefreshLookupList	public	TField
RemoveComponent	public	TComponent
SafeCallException	public	TComponent
SetData	public	TField
SetFieldType	public	TField
Validate	public	TField

Create

Create creates an instance of *TFloatField*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to create and initialize an instance of *TFloatField*. After calling the inherited constructor, *Create* sets the *DataType* to *ftFloat*, and the *Precision* to 15.

It is seldom necessary to call *Create* directly, because a floating-point field component is instantiated automatically for all floating-point fields in a dataset.

See also *DataType* property, *Precision* property, *TNumericField.Create* method

TFontDialog

Unit: Dialogs | Palette: Dialogs

TObject • TPersistent • TComponent • TCommonDialog • TFontDialog

TFontDialog displays a font-selection dialog.

Description *TFontDialog* displays a modal Windows dialog box for selecting fonts. The dialog does not appear at runtime until it is activated by a call to the *Execute* method. When the user selects a font and clicks OK, the dialog closes and the selected font is stored in the *Font* property.

Properties

The following table lists the properties of *TFontDialog*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Ctl3D	published	True	TCommonDialog
DesignInfo	public		TComponent
Device	published	fdScreen	TFontDialog
Font	published		TFontDialog
HelpContext	published	0	TCommonDialog
MaxFontSize	published		TFontDialog
MinFontSize	published		TFontDialog
Name	published		TComponent
Options	published	[fdEffects]	TFontDialog
Owner	public, read-only		TComponent
Tag	published	0	TComponent
VCLComObject	public		TComponent

Device

Selects the device from which to retrieve the list of available fonts.

```
type TFontDialogDevice = (fdScreen, fdPrinter, fdBoth);
property Device: TFontDialogDevice;
```

Description These are the possible values of *Device*:

Value	Meaning
fdScreen	Retrieve from the screen.
fdPrinter	Retrieve from the printer.
fdBoth	Retrieve from both the screen and the printer.

Font

Returns the selected font.

```
property Font: TFont;
```

Description When the user selects a font in the dialog box and clicks OK, the selected font becomes the value of the *Font* property. To make a default font appear in the dialog when it opens, assign a value to *Font* in the Object Inspector or in program code.

See also *TFont* object

MaxFontSize

Determines the largest font size available in the Font dialog.

```
property MaxFontSize: Integer;
```

Description Assign a value to *MaxFontSize* to limit the font sizes available in the dialog.

Note *MaxFontSize* is inoperative unless the *fdLimitSize* flag is set in *Options*.

See also *MinFontSize* property, *Options* property

MinFontSize

Determines the smallest font size available in the Font dialog.

```
property MinFontSize: Integer;
```

Description Assign a value to *MinFontSize* to limit the font sizes available in the dialog.

Note *MinFontSize* is inoperative unless the *fdLimitSize* flag is set in *Options*.

See also *MaxFontSize* property, *Options* property

Options

Determines the appearance and behavior of the Font dialog.

```
type
  TFontDialogOption = (fdAnsiOnly, fdTrueTypeOnly, fdEffects, fdFixedPitchOnly,
    fdForceFontExist, fdNoFaceSel, fdNoOEMFonts, fdNoSimulations, fdNoSizeSel,
    fdNoStyleSel, fdNoVectorFonts, fdShowHelp, fdWysiwyg, fdLimitSize, fdScalableOnly,
    fdApplyButton);
  TFontDialogOptions = set of TFontDialogOption;

property Options: TFontDialogOptions;
```

Description Use the *Options* property to customize the appearance and functionality of the dialog. Here are the possible values of *Options*:

Value	Meaning
fdAnsiOnly	Displays only fonts that use the Windows character set. Symbol fonts are not listed in the dialog.
fdApplyButton	Displays an Apply button in the dialog, whether or not there is an *OnApply* event handler.

Value	Meaning
fdEffects	Displays the Effects check boxes (Strikeout and Underline) and the Color list box in the dialog.
fdFixedPitchOnly	Displays only monospaced fonts in the dialog. Proportionally spaced fonts are not listed, nor are TrueType fonts that do not have the fixed-pitch flag set in the style header.
fdForceFontExist	Allows the user to enter only fonts that are displayed in the dialog (that is, listed in the Font combo box). If the user tries to enter another font name, an error message appears.
fdLimitSize	Enables the *MaxFontSize* and *MinFontSize* properties, limiting the range of font sizes that appear in the dialog if these properties have values.
fdNoFaceSel	Causes the dialog to open without a preselected font name in the Font combo box.
fdNoOEMFonts	Removes OEM fonts from the dialog's combo box. Lists only non-OEM fonts.
fdScalableOnly	Displays only scalable fonts in the dialog. Non-scalable (bitmap) fonts are removed from the list.
fdNoSimulations	Displays only fonts and font styles that are directly supported by the font definition file. GDI-synthesized bold and italic styles (for bitmap fonts) are not listed.
fdNoSizeSel	Causes the dialog to open without a preselected size in the Size combo box.
fdNoStyleSel	Causes the dialog to open without a preselected style in the Font Style combo box.
fdNoVectorFonts	Removes vector fonts from the dialog's combo box; lists only non-vector fonts. (Vector fonts are Windows 1.0 fonts, such as Roman or Script, which resemble pen-plotter output.)
fdShowHelp	Displays a Help button in the dialog.
fdTrueTypeOnly	Displays only TrueType fonts in the dialog. Other fonts are not listed.
fdWysiwyg	Displays only fonts that are available to both the printer and the screen. Device-specific fonts are not listed in the dialog.

By default, all options except *fdEffects* are off.

See also *OnApply* event, *MaxFontSize* property, *MinFontSize* property

Events

The following table lists the events of *TFontDialog*:

Event	Visibility	Documented in
OnApply	published	TFontDialog
OnClose	published	TCommonDialog
OnShow	published	TCommonDialog

OnApply

OnApply occurs when the user clicks the Apply button.

```
type TFDApplyEvent = procedure(Sender: TObject; Wnd: HWND) of object;

property OnApply: TFDApplyEvent;
```

Description The Apply button appears in the dialog if an event handler is assigned to *OnApply* or if the *fdApplyButton* flag is set in *Options*. Use an *OnApply* event handler to apply the selected font without closing the Font dialog box.

For example, the following event handler assigns the *Font* of a *TFontDialog* instance (*FontDialog1*) to the *Font* of a *TForm* instance (*Form1*).

```
procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: Integer);
begin
  Font.Assign(TFontDialog(Sender).Font);
end;
```

Note The *Wnd* parameter is the Font dialog's window handle. *Wnd* has the same value as the *Handle* property and is maintained for compatibility with previous versions of Delphi.

See also *Font* property, *Handle* property, *Options* property, *TForm.Font* property

Methods

The following table lists the methods of *TFontDialog*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TFontDialog
DefaultHandler	public	TObject
Destroy	public	TFontDialog
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
Execute	public	TFontDialog
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject

Method	Visibility	Documented in
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Create

Creates and initializes a *TFontDialog* instance.

constructor Create(AOwner: TComponent); **override**;

Description The *Create* method generates a *TFontDialog* instance, but the new dialog does not appear on the form at runtime until the *Execute* method is called.

See also *Execute* method

Destroy

Destroys the *TFontDialog* and frees its memory.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Call *Free* instead. *Free* verifies that the Font dialog is not already destroyed before calling *Destroy*.

See also *Free* method

Execute

Displays the Font dialog.

function Execute: Boolean; **override**;

Description *Execute* opens the Font dialog, returning *True* when the user selects a font and clicks OK.

TForm

Unit: Forms

TObject • TPersistent • TComponent • TControl • TWinControl • TScrollingWinControl • TCustomForm • TForm

TForm is the component for a standard application window (form).

Description Use *TForm* to create a form, such as a main window, dialog box, or MDI child. The form can contain other objects, such as *TButton*, *TCheckBox*, and *TComboBox* objects.

Examples of forms include *TLoginDialog* and *TPasswordDialog* objects.

See also *TLoginDialog* object, *TPasswordDialog* object

Properties

The following table lists the properties of *TForm*:

Property	Visibility	Default	Documented in
Active	public, read-only		TCustomForm
ActiveControl	published		TCustomForm
ActiveMDIChild	public, read-only		TCustomForm
ActiveOleControl	public		TCustomForm
Align	public	alNone	TControl
AutoScroll	published	True	TScrollingWinControl
BorderIcons	published	[biSystemMenu, biMinimize, biMaximize]	TCustomForm
BorderStyle	published	bsSizeable	TCustomForm
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Canvas	public, read-only		TCustomForm
Caption	published		TControl
ClientHandle	public, read-only		TCustomForm
ClientHeight	published		TCustomForm
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TCustomForm
ClientWidth	published		TCustomForm
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent

Property	Visibility	Default	Documented in
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published	True	TWinControl
Cursor	published	crDefault	TControl
Designer	public		TCustomForm
DesignInfo	public		TComponent
DropTarget	public		TCustomForm
Enabled	published	True	TControl
Font	published		TControl
FormStyle	published	fsNormal	TCustomForm
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
HelpFile	published		TCustomForm
Hint	published		TControl
HorzScrollBar	published		TScrollingWinControl
Icon	published		TCustomForm
KeyPreview	published	False	TCustomForm
Left	published		TControl
MDIChildCount	public, read-only		TCustomForm
MDIChildren	public, read-only		TCustomForm
Menu	published		TCustomForm
ModalResult	public		TCustomForm
Name	published		TControl
ObjectMenuItem	published		TCustomForm
OleFormObject	public		TCustomForm
Owner	public, read-only		TComponent
Parent	public		TCustomForm
ParentFont	published	False	TControl
ParentWindow	public		TWinControl
PixelsPerInch	published		TCustomForm
PopupMenu	published		TControl
Position	published	poDesigned	TCustomForm
PrintScale	published	poProportional	TCustomForm
Scaled	published	True	TCustomForm
ShowHint	published		TControl

Property	Visibility	Default	Documented in
Showing	public, read-only		TWinControl
TabOrder	public	–1	TWinControl
TabStop	public	False	TWinControl
Tag	published	0	TComponent
TileMode	public	tbHorizontal	TCustomForm
Top	published		TControl
VCLComObject	public		TComponent
VertScrollBar	published		TScrollingWinControl
Visible	published	False	TCustomForm
Width	published		TControl
WindowMenu	published		TCustomForm
WindowProc	public		TControl
WindowState	published	wsNormal	TCustomForm

Events

The following table lists the events of *TForm*:

Event	Visibility	Documented in
OnActivate	published	TCustomForm
OnClick	published	TControl
OnClose	published	TCustomForm
OnCloseQuery	published	TCustomForm
OnCreate	published	TCustomForm
OnDblClick	published	TControl
OnDeactivate	published	TCustomForm
OnDestroy	published	TCustomForm
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnHelp	published	TCustomForm
OnHide	published	TCustomForm
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnPaint	published	TCustomForm
OnResize	published	TCustomForm
OnShow	published	TCustomForm

Methods

The following table lists the methods of *TForm*:

Method	Visibility	Documented in
ArrangeIcons	public	TForm
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
Cascade	public	TForm
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
Close	public	TCustomForm
CloseQuery	public	TCustomForm
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomForm
CreateNew	public	TCustomForm
CreateParented	public	TWinControl
DefaultHandler	public	TObject
DefocusControl	public	TCustomForm
Destroy	public	TCustomForm
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
DisableAutoRange	public	TScrollingWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EnableAutoRange	public	TScrollingWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
FocusControl	public	TCustomForm
Focused	public	TWinControl

Method	Visibility	Documented in
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetFormImage	public	TCustomForm
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TCustomForm
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Next	public	TForm
PaintTo	public	TWinControl
Perform	public	TControl
Previous	public	TForm
Print	public	TCustomForm
Realign	public	TWinControl
Refresh	public	TControl
Release	public	TCustomForm
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
ScrollInView	public	TScrollingWinControl

Method	Visibility	Documented in
SendCancelMode	public	TCustomForm
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TCustomForm
SetFocusedControl	public	TCustomForm
SetTextBuf	public	TControl
Show	public	TCustomForm
ShowModal	public	TCustomForm
Tile	public	TForm
Update	public	TWinControl
UpdateControlState	public	TWinControl

ArrangeIcons

ArrangeIcons arranges the icons of minimized MDI child forms.

procedure ArrangeIcons;

Description Use *ArrangeIcons* to arrange the icons of minimized MDI child forms so they are evenly spaced and don't overlap.

ArrangeIcons works only if the form is an MDI parent form (that is, if the form's *FormStyle* property is *fsMDIForm*).

See also *Cascade* method, *Tile* method, *FormStyle* property

Cascade

Cascade arranges MDI child forms so they overlap.

procedure Cascade;

Description Use *Cascade* to arrange MDI child forms so they overlap.

Cascade works only if the form is an MDI parent form (that is, if the form's *FormStyle* property is *fsMDIForm*).

See also *Tile* method, *ArrangeIcons* method, *FormStyle* property

Next

The *Next* method makes the next child form in the form sequence the active form.

procedure Next;

Description For example, if you have three child forms within a parent form in your MDI application and Form2 is the active form, the *Next* method makes Form3 the active form. Calling *Next* again makes Form4 active. The next time your

application calls *Next*, the sequence starts over again and Form2 becomes the active form once again.

The *Next* method applies only to forms that are MDI parent forms (have a *FormStyle* property value of *fsMDIForm*).

See also *Previous* method, *FormStyle* property

Previous

The *Previous* method makes the previous MDI child form in the form sequence the active form.

```
procedure Previous;
```

Description For example, if you have three child forms within a parent form in your MDI application and Form4 is the active form, the *Previous* method makes Form3 the active form. Calling *Previous* again makes Form2 active. The next time your application calls *Previous*, the sequence starts over again and Form4 becomes the active form once again.

The *Previous* method applies only to forms that are MDI parent forms (have a *FormStyle* property value of *fsMDIForm*).

See also *Next* method

Tile

Tile arranges MDI child forms so that they are all the same size.

```
procedure Tile;
```

Description Use *Tile* to arrange MDI child forms so that they are all the same size. Tiled forms completely fill up the client area of the parent form. How the forms arrange themselves depends upon the values of their *TileMode* properties.

Tile works only if the form is an MDI parent form (that is, if the form's *FormStyle* property is *fsMDIForm*).

See also *TileMode* property

TGraphicControl

Unit: Controls

TObject • TPersistent • TComponent • TControl • TGraphicControl

TGraphicControl is the base class for all nonwindowed custom controls.

Description The primary purpose of most graphic controls is to display text or graphics. Delphi controls that descend from *TGraphicControl* include *TBevel*, *TImage*,

TPaintBox, *TShape*, *TSpeedButton*, *TSplitter*, and *TCustomLabel*, from which *TDBText* and *TLabel* are derived.

In addition to everything inherited from the abstract *TControl* component, *TGraphicControl* provides a *Canvas* property that provides ready access to the control's drawing surface and a virtual *Paint* method called in response to *WM_PAINT* messages received by a parent control.

Graphic controls cannot receive the input focus, nor can they contain other controls. Because a graphic control doesn't need a Windows handle, its demand on system resources is lessened, and painting a graphic control is quicker than painting a windowed control.

To create a control that can receive input focus or contain other controls, but which needs a *Canvas* property and a *Paint* method, derive a class from *TCustomControl*.

See also *TBevel* object, *TCustomControl* object, *TCustomLabel* object, *TImage* object, *TPaintBox* object, *TShape* object, *TSpeedButton* object, *TSplitter* object

Properties

The following table lists the properties of *TGraphicControl*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
BoundsRect	public		TControl
Canvas	protected, read-only		TGraphicControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TControl
ClientRect	public, read-only		TControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
Enabled	public	True	TControl
Height	published		TControl
Hint	published		TControl
Left	published		TControl
Name	published		TControl

Property	Visibility	Default	Documented in
Owner	public, read-only		TComponent
Parent	public		TControl
ShowHint	public		TControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	public	True	TControl
Width	published		TControl
WindowProc	public		TControl

Canvas

Canvas is a *TCanvas* object that is used as a drawing surface by the graphic control.

```
property Canvas: TCanvas;
```

Description Use the properties of the *TCanvas* object to draw or paint on the surface of the control. *Canvas* provides all the tools and methods needed for drawing and painting.

Since *TGraphicControl* does not have a window handle, when *Canvas* needs a handle to a device context, it uses the device context obtained from the window of the control's *Parent*.

See also *Paint* method, *TCanvas* object, *TControl.GetDeviceContext* method, *TControl.Parent* property

Methods

The following table lists the methods of *TGraphicControl*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
Create	public	TGraphicControl
DefaultHandler	public	TObject

Method	Visibility	Documented in
Destroy	public	TGraphicControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTextBuf	public	TControl
GetTextLen	public	TControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
Invalidate	public	TControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Paint	protected	TGraphicControl
Perform	public	TControl
Refresh	public	TControl
RemoveComponent	public	TComponent
Repaint	public	TControl
SafeCallException	public	TComponent
ScreenToClient	public	TControl
SendToBack	public	TControl
SetBounds	public	TControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TControl

Create

Create creates an instance of *TGraphicControl*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to instantiate and initialize a graphic control at runtime. Graphic controls placed on forms at design time are created automatically.

Create creates the *TCanvas* object for the *Canvas* property after calling the inherited constructor.

Nearly every kind of control overrides *Create* to initialize its unique properties. When overriding the *Create* method, always call the inherited *constructor* first, then proceed with the component's initialization.

See also *Canvas* property, *TControl.Create* method

Destroy

Destroy destroys an instance of *TGraphicControl*.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the graphic control is not already freed, and only then calls *Destroy*.

Use *Destroy* to free any memory or resources allocated in the *Create* method. When declaring a *Destroy* method in a descendant control type, always add the **override** directive to the declaration and call the inherited *Destroy* as the last statement in the redeclared method.

See also *Create* method, *Free* method

Paint

Paint provides the interface for a method that renders the image of the graphic control.

procedure Paint; **virtual**;

Description A graphic control receives WM_PAINT messages from its *Parent*, when the parent wants the graphic control to paint itself. Graphic controls respond to *WM_PAINT* messages by initializing the control's canvas and calling the *Paint* method. Override *Paint* to specify how the image of the control should be drawn.

The *Paint* method for *TGraphicControl* does nothing more than provide an interface to be overridden by derived objects. All graphic controls that can be instantiated must override *Paint* to draw the image of the control.

See also *Parent* property, *TWinControl.PaintControls* method

TGraphicField

Unit: DB

TObject • TPersistent • TComponent • TField • TBlobField • TGraphicField

The *TGraphicField* component represents a graphics field in a dataset.

Description Graphics fields are a form of binary large object (BLOB) field where the data includes a BLOB header describing the encoding of the graphical value. *TGraphicField* differs from its immediate ancestor *TBlobField* only in having a *DataType* of *ftGraphic*. As a descendant of *TBlobField*, *TGraphicField* includes many properties, methods, and events that are useful for managing the value and properties of a BLOB field in a database.

See also *TImage* object, *TMemoField* object, *TPicture* object

Properties

The following table lists the properties of *TGraphicField*:

Property	Visibility	Default	Documented in
Alignment	published	taLeftJustify	TField
AsBoolean	public		TField
AsCurrency	public		TField
AsDateTime	public		TField
AsFloat	public		TField
AsInteger	public		TField
AsString	public		TBlobField
AsVariant	public		TBlobField
AttributeSet	public		TField
BlobSize	public, read-only		TBlobField
BlobType	published		TBlobField
Calculated	public	False	TField
CanModify	public, read-only		TField
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ConstraintErrorMessage	published		TField
CurValue	public, read-only		TField
CustomConstraint	published		TField
DataSet	public		TField

Property	Visibility	Default	Documented in
DataSize	public, read-only		TField
DataType	public, read-only		TField
DefaultExpression	published		TField
DesignInfo	public		TComponent
DisplayLabel	published		TField
DisplayName	public, read-only		TField
DisplayText	public, read-only		TField
DisplayWidth	published		TField
EditMask	public		TField
EditMaskPtr	public, read-only		TField
FieldKind	published	fkData	TField
FieldName	published		TField
FieldNo	public, read-only		TField
HasConstraints	published, read-only		TField
ImportedConstraint	published		TField
Index	published		TField
IsIndexField	public, read-only		TField
IsNull	public, read-only		TBlobField
KeyFields	published		TField
Lookup	public		TField
LookupCache	published	False	TField
LookupDataSet	published		TField
LookupKeyFields	published		TField
LookupList	public, read-only		TField
LookupResultField	published		TField
Modified	public		TBlobField
Name	published		TComponent
NewValue	public		TField
Offset	public, read-only		TField
OldValue	public, read-only		TField
Origin	published		TField
Owner	public, read-only		TComponent
ReadOnly	published	False	TField
Required	published	False	TField
Size	published	0	TField
Tag	published	0	TComponent
Text	public		TField
Transliterate	public		TBlobField
ValidChars	public		TField
Value	public		TBlobField
VCLComObject	public		TComponent
Visible	published	True	TField

Events

The following table lists the events of *TGraphicField*:

Event	Visibility	Documented in
OnChange	published	TField
OnGetText	published	TField
OnSetText	published	TField
OnValidate	published	TField

Methods

The following table lists the methods of *TGraphicField*:

Method	Visibility	Documented in
Assign	public	TBlobField
AssignValue	public	TField
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TBlobField
Create	public	TGraphicField
DefaultHandler	public	TObject
Destroy	public	TField
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
FocusControl	public	TField
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetData	public	TField
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent

Method	Visibility	Documented in
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
IsBlob	public	TBlobField
IsValidChar	public	TField
LoadFromFile	public	TBlobField
LoadFromStream	public	TBlobField
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RefreshLookupList	public	TField
RemoveComponent	public	TComponent
SafeCallException	public	TComponent
SaveToFile	public	TBlobField
SaveToStream	public	TBlobField
SetData	public	TField
SetFieldType	public	TBlobField
SetText	public	TBlobField
Validate	public	TField

Create

Create is the constructor for a graphic field component.

constructor Create(AOwner: TComponent); **override**;

Description The *Create* method creates and initializes an instance of *TGraphicField*. After calling the inherited constructor, *Create* sets the *DataType* to *ftGraphic*.

It is seldom necessary to call *Create* because a graphic field component is instantiated automatically for all graphic fields in a dataset.

See also *DataType* property, *TField.Create* method

TGroupBox

Unit: StdCtrls | Palette: Standard

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomControl • TCustomGroupBox • TGroupBox

TGroupBox creates a Windows group box.

Description The *TGroupBox* component represents a standard Windows group box, used to group related controls on a form. When another control component is placed within a group box, the group box becomes the parent of that component.

See also *TRadioGroup* object, *TDBRadioGroup* object, *Parent* property

Properties

The following table lists the properties of *TGroupBox*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Caption	published		TControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
Name	published		TControl

Property	Visibility	Default	Documented in
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	True	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Events

The following table lists the events of *TGroupBox*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TGroupBox*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomGroupBox
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TCustomControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent

Method	Visibility	Documented in
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

THeaderControl

Unit: ComCtrls | Palette: Win32

TObject • TPersistent • TComponent • TControl • TWinControl • THeaderControl

THeaderControl is a container for *THeaderSection* objects. It provides a set of resizable column headers.

Description A *THeaderControl* is a set of column headers that the user can resize at runtime. Each header is represented by a *THeaderSection* object listed in the *Sections* property.

The header sections can be positioned above columns or fields of information. For example, header sections could be placed over a list box (*TListBox*). Because *THeaderControl* provides a generic header element that can be paired with virtually any other visual component, runtime manipulation of the header sections is not automatically extended to the components beneath it; if, for example, you want the column beneath a header section to be resized whenever the header section is resized, you will have to write a custom *OnSectionResize* event handler to accomplish this.

See also *THeaderSection* object, *THeaderSections* object, *THeader* object, *TListBox* object

Properties

The following table lists the properties of *THeaderControl*:

Property	Visibility	Default	Documented in
Align	published	alTop	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Canvas	public, read-only		THeaderControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl

Property	Visibility	Default	Documented in
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
HotTrack	published	False	THeaderControl
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
Sections	published		THeaderControl
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	public	–1	TWinControl
TabStop	public	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Canvas

Provides runtime access to the header control's drawing surface.

property Canvas: TCanvas;

Description The read-only *Canvas* property provides access to the header control's drawing surface that can be used when implementing a handler for the *OnDrawSection* event.

See also *TCanvas* object, *OnDrawSection* event

HotTrack

Highlights the header sections as the mouse passes over them.

property HotTrack: Boolean;

Description If *HotTrack* is set to *True*, the text on each header section appears highlighted when the mouse passes over it at runtime.

See also *THeaderSection.Text* property

Sections

A list of header sections (column headings).

property Sections: THeaderSections;

Description The *Sections* property holds a *THeaderSections*—that is, a collection of *THeaderSection* objects. At design time, header sections can be added, removed, or modified with the Sections editor. To open the Sections editor, select the *Sections* property in the Object Inspector, then double-click in the Value column to the right or click the ellipsis (...) button.

See also *THeaderSection* object, *THeaderSections* object

Events

The following table lists the events of *THeaderControl*:

Event	Visibility	Documented in
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnDrawSection	published	THeaderControl
OnEndDrag	published	TControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnResize	published	THeaderControl
OnSectionClick	published	THeaderControl
OnSectionResize	published	THeaderControl
OnSectionTrack	published	THeaderControl
OnStartDrag	published	TControl

OnDrawSection

OnDrawSection occurs when a header section needs to be redrawn.

type TDrawSectionEvent = **procedure** (HeaderControl: THeaderControl;

```
Section: THeaderSection; const Rect: TRect; Pressed: Boolean) of object;
property OnDrawSection: TDrawSectionEvent;
```

Description The *OnDrawSection* event occurs when a header section needs to be redisplayed—for example, when the user selects or resizes a section. *OnDrawSection* occurs only if the header section's *Style* property is set to *hsOwnerDraw*.

See also *THeaderSection.Style* property

OnResize

OnResize occurs when the header control is resized.

```
type TNotifyEvent = procedure(Sender: TObject) of object;
property OnResize: TNotifyEvent;
```

Description The *OnResize* event occurs when the size of the header control changes at runtime.

See also *TNotifyEvent* type

OnSectionClick

OnSectionClick occurs when one of the header control's sections is clicked.

```
type TSectionNotifyEvent = procedure(HeaderControl: THeaderControl;
  Section: THeaderSection) of object;
property OnSectionClick: TSectionNotifyEvent;
```

Description The *OnSectionClick* event occurs when a header section is clicked at runtime. To disable a section from being clicked set its *AllowClick* property to *False*.

TSectionNotifyEvent is the type of the *OnSectionClick* and *OnSectionResize* events.

See also *THeaderSection.AllowClick* property

OnSectionResize

OnSectionResize occurs when one of the header control's sections is resized.

```
type TSectionNotifyEvent = procedure(HeaderControl: THeaderControl;
  Section: THeaderSection) of object;
property OnSectionResize: TSectionNotifyEvent;
```

Description The *OnSectionResize* event occurs when a header section is resized at runtime. This happens when the user positions the mouse pointer between two sections and drags their border to the right or left.

TSectionNotifyEvent is the type of the *OnSectionClick* and *OnSectionResize* events.

See also *OnSectionTrack* event

OnSectionTrack

OnSectionTrack occurs as one of the header control's sections is being resized.

```
type
  TSectionTrackState = (tsTrackBegin, tsTrackMove, tsTrackEnd);
  TSectionTrackEvent = procedure(HeaderControl: THeaderControl; Section: THeaderSection;
    Width: Integer; State: TSectionTrackState) of object;

property OnSectionTrack: TSectionTrackEvent;
```

Description The *OnSectionTrack* event tracks the dragging of a header section's border as it happens. *OnSectionTrack* occurs when the mouse pointer is positioned between two header sections and the left mouse button is pressed.

OnSectionTrack is of type *TSectionTrackEvent*, whose *Width* and *State* parameters indicate the size of the header section (in pixels) and the status of the event. The *State* property can take three values:

Value	Meaning
tsTrackBegin	The border has not yet been dragged.
tsTrackMove	The border is being dragged.
tsTrackEnd	The border is no longer being dragged.

Use *OnSectionTrack* when you want something to happen at runtime as a header section is being resized.

See also *OnSectionResize* event

Methods

The following table lists the methods of *THeaderControl*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject

Method	Visibility	Documented in
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	THeaderControl
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	THeaderControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject

Method	Visibility	Documented in
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

Create

Creates and initializes a *THeaderControl* component.

constructor Create(AOwner: TComponent); **override**;

Description The *Create* method creates a header control along with its canvas and header sections.

See also *TCanvas* object, *THeaderSections* object

Destroy

Destroys *THeaderControl* and frees its memory.

destructor Destroy; **override**;

Description The *Destroy* method destroys the header control along with its canvas and header sections.

See also *TCanvas* object, *THeaderSections* object

THintWindow

Unit: Controls

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomControl • THintWindow

THintWindow implements the small pop-up window that appears over a control at runtime when the control has its *ShowHint* property set to *True*.

Description Use *THintWindow* to display a Help Hint pop-up window directly from an application. Override *THintWindow* to customize the window that appears automatically for controls with their *ShowHint* property set to *True*. After overriding *THintWindow* to create a new derived type, assign the new type to the global *HintWindowClass* variable at application startup, so that the new hint window type is used for Help Hints.

See also *HintWindowClass* variable, *TApplication.Hint* property

Properties

The following table lists the properties of *THintWindow*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Canvas	public, read-only		TCustomControl
Caption	public		TControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	public	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent

Property	Visibility	Default	Documented in
Enabled	public	True	TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentWindow	public		TWinControl
ShowHint	public		TControl
Showing	public, read-only		TWinControl
TabOrder	public	–1	TWinControl
TabStop	public	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	public	True	TControl
Width	published		TControl
WindowProc	public		TControl

Methods

The following table lists the methods of *THintWindow*:

Method	Visibility	Documented in
ActivateHint	public	THintWindow
ActivateHintData	public	THintWindow
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CalcHintRect	public	THintWindow
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl

Method	Visibility	Documented in
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	THintWindow
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TCustomControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
IsHintMsg	public	THintWindow
MethodAddress	public	TObject
MethodName	public	TObject

Method	Visibility	Documented in
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
ReleaseHandle	public	THintWindow
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

ActivateHint

ActivateHint displays the hint window.

```
procedure ActivateHint(Rect: TRect; const AHint: string); virtual;
```

Description Call *ActivateHint* to display the hint window at the coordinates provided by the *Rect* parameter. If the *Rect* parameter appears off screen, *ActivateHint* moves the hint window to the closest approximation to the coordinates supplied by *Rect* that appears entirely on screen. *ActivateHint* sets the *Caption* property to the *AHint* parameter before showing the window, so that the *AHint* string appears to the user.

After an application displays a hint window, the application automatically deactivates it after a time period has elapsed, or when it receives a message that requires hiding the hint message. To hide the hint window that is brought up automatically by the application, use the *HideHint* method of the global *Application* object. To hide a hint window that is activated directly by calling *ActivateHint*, use the *ReleaseHandle* method.

See also *Application* variable, *IsHintMsg* method, *ReleaseHandle* method, *TApplication.HideHint* method, *TControl.Caption* property

ActivateHintData

ActivateHintData displays the hint window using a provided data value.

```
procedure ActivateHintData(Rect: TRect; const AHint: string; AData: Pointer); virtual;
```

Description Call *ActivateHintData* to display the hint window at the coordinates provided by the *Rect* parameter. The *ActivateHintData* method for *THintWindow* sets the *Caption* property to the *AHint* parameter and ignores the *AData* parameter. Descendant classes can override this method to combine the *AHint* and *AData* parameters into a customized hint value.

After an application displays a hint window, it automatically deactivates it after a time period has elapsed, or when it receives a message that requires hiding the hint message. To hide the hint window that is brought up automatically by the application, use the *HideHint* method of the global *Application* object. To hide a hint window that is activated directly by calling *ActivateHint*, use the *ReleaseHandle* method.

See also *ActivateHint* method, *Application* variable, *IsHintMsg* method, *ReleaseHandle* method, *TApplication.HideHint* method, *TControl.Caption* property

CalcHintRect

CalcHintRect returns the size that is needed to display a given hint.

```
function CalcHintRect(MaxWidth: Integer; const AHint: string; AData: Pointer): TRect; virtual;
```

Description Call *CalcHintRect* to determine the size needed to display a hint window for the string and data supplied by the *AHint* and *AData* parameters. The *MaxWidth* parameter limits how wide the hint window can grow before the text wraps to a new line.

See also *ActivateHintData* method, *TRect* type

Create

Create creates and initializes an instance of *THintWindow*.

```
constructor Create(AOwner: TComponent); override;
```

Description Call *Create* to instantiate a hint window at runtime. Hint windows activated by the application in response to the *ShowHint* property of a control are created automatically.

See also *ActivateHint* method, *TControl.ShowHint* property

IsHintMsg

IsHintMsg determines whether an application message requires hiding the hint window.

```
function IsHintMsg(var Msg: TMsg): Boolean; virtual;
```

Description Call *IsHintMsg* to determine whether the message specified by the *Msg* parameter requires hiding the hint window. Upon seeing a mouse, keyboard, command, or activation message, *IsHintMsg* returns *True*. The global *Application* object calls *IsHintMsg* to check messages while the hint window is on the screen, and hides the window if *IsHintMsg* returns *True*.

See also *Application* variable

ReleaseHandle

ReleaseHandle destroys the windows displayed by *THintWindow*.

```
procedure ReleaseHandle;
```

Description Call *ReleaseHandle* to destroy a hint window that was activated by directly calling the *ActivateHint* method. Hint windows activated automatically by the application object are deactivated by the application as well.

See also *ActivateHint* method

THotKey

Unit: ComCtrls — Palette: Win32

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomHotKey • THotKey

THotKey is a wrapper for a hot key Windows Common Control.

Description *THotKey* implements the generic behavior introduced in *TCustomHotKey*. *THotKey* publishes many of the properties inherited from *TCustomHotKey*, but does not introduce any new behavior.

Use *THotKey* to create key combinations that can be used as a shortcut.

Properties

The following table lists the properties of *THotKey*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
AutoSize	published	True	TCustomHotKey
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent

Property	Visibility	Default	Documented in
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
Enabled	published	True	TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
HotKey	published		TCustomHotKey
InvalidKeys	published		TCustomHotKey
Left	published		TControl
Modifiers	published		TCustomHotKey
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Events

The following table lists the events of *THotKey*:

Event	Visibility	Documented in
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl

Methods

The following table lists the methods of *THotKey*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomHotKey
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TWinControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl

Method	Visibility	Documented in
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl

Method	Visibility	Documented in
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

TImage

Unit: ExtCtrls

Palette: Additional

TObject • TPersistent • TComponent • TControl • TGraphicControl • TImage

TImage displays a graphical image.

Description

Use *TImage* to display a graphical image on a form. Use a *TPicture* object to specify the actual bitmap, icon, metafile, or other graphic object displayed by *TImage*. *TImage* introduces several properties to determine how the image is displayed within the boundaries of the *TImage* object.

To add an image to a form or data module so that it is available for display by other controls, use *TImageList* instead.

See also

TImageList object, *TPicture* object

Properties

The following table lists the properties of *TImage*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
AutoSize	published	False	TImage
BoundsRect	public		TControl
Canvas	public, read-only		TImage
Center	published	False	TImage
ClientHeight	public		TControl
ClientOrigin	public, read-only		TControl
ClientRect	public, read-only		TControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlState	public		TControl

Property	Visibility	Default	Documented in
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Height	published		TControl
Hint	published		TControl
IncrementalDisplay	published	False	TImage
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentShowHint	published	True	TControl
Picture	published		TImage
PopupMenu	published		TControl
ShowHint	published		TControl
Stretch	published	False	TImage
Tag	published	0	TComponent
Top	published		TControl
Transparent	published	False	TImage
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

AutoSize

AutoSize determines whether the image control resizes to accommodate the image it displays.

property AutoSize: Boolean;

Description Set *AutoSize* to *True* to ensure that the entire image is visible within the image control. When *AutoSize* is *True*, the image control resizes to accommodate the image specified by the *Picture* property. When *AutoSize* is *False*, the image control remains the same size, regardless of the size of the image. If the image control is smaller than the image, only the portion of the picture that fits inside the image control is visible.

To resize the image to fit the image control rather than resizing the control to fit the image, use the *Stretch* property.

See also *BoundsRect* property, *Picture* property, *Stretch* property

Canvas

Canvas is the drawing surface used to display the image.

property Canvas: TCanvas;

Description Use *Canvas* to add to the image specified by the *Picture* property. For example, use the *TextOut* method of the *TCanvas* object to add labels to portions of the image.

Note *Canvas* is only available if the *Picture* property represents a bitmap. Trying to read *Canvas* when *Picture* is another type of graphic image raises an *EInvalidOperation* exception.

See also *EInvalidOperation* object, *Picture* property, *TCanvas* object

Center

Center positions the image so that it is centered in the image control.

property Center: Boolean;

Description When the image does not fit perfectly within the image control, use *Center* to specify how the image is positioned. When *Center* is *True*, the image is centered in the control. When *Center* is *False*, the upper left corner of the image is positioned at the upper left corner of the control.

Note *Center* has no effect if the *AutoSize* property is *True* or if the *Stretch* property is *True* and *Picture* does not specify an icon.

See also *AutoSize* property, *Picture* property, *Stretch* property

IncrementalDisplay

IncrementalDisplay specifies whether successive approximations of the image should be drawn during slow operations.

property IncrementalDisplay: Boolean;

Description Set *IncrementalDisplay* to *True* on large compressed images that take a long time to load or on images that may be altered by a computationally slow process. When *IncrementalDisplay* is *True*, partial renderings of the image are drawn periodically during slow operations such as loading. Set *IncrementalDisplay* to *False* to wait until the entire operation is complete before drawing the image.

For some classes of graphic objects, an *OnProgress* event occurs at various points during slow operations. If *IncrementalDisplay* is *True*, the *OnProgress* event occurs immediately after the approximate image is drawn. If *IncrementalDisplay* is *False*, *OnProgress* occurs, but the image is not updated.

See also *OnProgress* event

Picture

Picture specifies the image that appears on the image control.

property Picture: TPicture;

Description Set *Picture* to a *TPicture* object that contains an icon, metafile, bitmap graphic, or user-defined graphic to be displayed by the image control. Setting *Picture* at design time brings up the Picture Editor, which can be used to specify the file that contains the image.

See also *TPicture* object

Stretch

Stretch resizes the image specified by the *Picture* property so that it exactly fits the bounds of the image control.

property Stretch: Boolean;

Description Set *Stretch* to *True* to cause the image to assume the size and shape of the image control. When the image control resizes, the image resizes also. *Stretch* resizes the height and width of the image independently. Thus, unlike a simple change in magnification, stretch can distort the image if the image control is not the same shape as the image.

To resize the control to the image rather than resizing the image to the control, use the *AutoSize* property instead.

Note *Stretch* has no effect if the *Picture* property contains an icon.

See also *AutoSize* property, *Picture* property

Transparent

Transparent specifies whether the background of the image obscures objects below the image object.

property Transparent: Boolean;

Description Set *Transparent* to *True* to allow objects behind the *TImage* object to show through the background of the bitmap. Set *Transparent* to *False* to make the background of the bitmap opaque. Setting *Transparent* sets the *Transparent* property of the *Picture*.

Note *Transparent* has no effect unless the *Picture* property specifies a bitmap.

See also *Picture* property, *TGraphic.Transparent* property

Events

The following table lists the events of *TImage*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnProgress	published	TImage
OnStartDrag	published	TControl

OnProgress

OnProgress occurs periodically during slow operations that affect the image.

```
type
  TProgressStage = (psStarting, psRunning, psEnding);
  TProgressEvent = procedure (Sender: TObject; Stage: TProgressStage;
    PercentDone: Byte; RedrawNow: Boolean; const R: TRect; const Msg: string) of object;

property OnProgress: TProgressEvent
```

Description

Write an *OnProgress* event handler to provide the user with feedback during slow operations such as loading large compressed images.

The *Stage* parameter indicates whether the operation is beginning, continuing, or ending. If the *OnProgress* event handler displays an indicator such as a progress bar, the indicator can be created when *Stage* is *psStarting*, updated while *State* is *psRunning*, and removed when *State* is *psEnding*. The *PercentDone* parameter is an approximation of how much of the operation has completed. Use *PercentDone* to update the position of a progress bar or other indicator.

The *RedrawNow* parameter indicates whether the image can safely be drawn on screen at this point. If the *IncrementalDisplay* property is *True*, an approximate image is drawn immediately before the *OnProgress* event when *RedrawNow* is *True*. The *R* parameter specifies the portion of the image that has changed and needs to be redrawn.

The *Msg* parameter contains one or two words that describe what operation is occurring. For example, the value of *Msg* could be a string such as *Loading*, *Storing*, or *Reducing colors*. The *Msg* string can also be empty.

Note The *PercentDone* parameter is only an approximation. With some image formats, the value of *PercentDone* may actually decrease from the value in previous events, as the graphic object discovers there is more work to do.

See also *IncrementalDisplay* property, *TPicture.OnProgress* event, *TProgressBar* object

Methods

The following table lists the methods of *TImage*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
Create	public	TImage
DefaultHandler	public	TObject
Destroy	public	TImage
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTextBuf	public	TControl
GetTextLen	public	TControl
HasParent	public	TComponent

Method	Visibility	Documented in
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
Invalidate	public	TControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Perform	public	TControl
Refresh	public	TControl
RemoveComponent	public	TComponent
Repaint	public	TControl
SafeCallException	public	TComponent
ScreenToClient	public	TControl
SendToBack	public	TControl
SetBounds	public	TControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TControl

Create

Create creates and initializes an instance of *TImage*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to instantiate a *TImage* object at runtime. Images placed on forms at design time are created automatically.

See also *TGraphicControl.Create* method

Destroy

Destroy destroys an instance of *TImage*.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the image is not already freed, and only then calls *Destroy*.

See also *Free* method

TImageList

Unit: Controls | Palette: Win32

TObject • TPersistent • TComponent • TCustomImageList • TImageList

The *TImageList* component is a collection of same-sized images, each of which can be referred to by its index.

Description Image lists are used to efficiently manage large sets of icons or bitmaps. All images in an image list are contained in a single, wide bitmap in screen device format. An image list may also include a monochrome bitmap that contains masks used to draw images transparently (icon style).

The image list is capable of holding a large number of same sized images and retrieving them via their index within the range 0 to n–1. The image list also has methods to facilitate storing, retrieving, and drawing of the stored images.

The *TImageList* is derived from *TCustomImageList* and exposes many of the protected properties and methods of the *TCustomImageList*.

See also *TListView* object, *TImage* object

Properties

The following table lists the properties of *TImageList*:

Property	Visibility	Default	Documented in
AllocBy	published	4	TCustomImageList
BkColor	published	clNone	TCustomImageList
BlendColor	published	clNone	TCustomImageList
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Count	public, read-only		TCustomImageList
DesignInfo	public		TComponent
DragCursor	public		TCustomImageList
Dragging	public, read-only		TCustomImageList
DrawingStyle	published	dsNormal	TCustomImageList
Handle	public		TCustomImageList
Height	published	16	TCustomImageList
ImageType	published	itImage	TCustomImageList
Masked	published	True	TCustomImageList

Property	Visibility	Default	Documented in
Name	published		TComponent
Owner	public, read-only		TComponent
ShareImages	published	False	TCustomImageList
Tag	published	0	TComponent
VCLComObject	public		TComponent
Width	published	16	TCustomImageList

Events

The following table lists the events of *TImageList*:

Event	Visibility	Documented in
OnChange	published	TCustomImageList

Methods

The following table lists the methods of *TImageList*:

Method	Visibility	Documented in
Add	public	TCustomImageList
AddIcon	public	TCustomImageList
AddImages	public	TCustomImageList
AddMasked	public	TCustomImageList
Assign	public	TCustomImageList
BeginDrag	public	TCustomImageList
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TCustomImageList
Create	public	TCustomImageList
CreateSize	public	TCustomImageList
DefaultHandler	public	TObject
Delete	public	TCustomImageList
Destroy	public	TCustomImageList
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
DragLock	public	TCustomImageList

Method	Visibility	Documented in
DragMove	public	TCustomImageList
DragUnlock	public	TCustomImageList
Draw	public	TCustomImageList
DrawOverlay	public	TCustomImageList
EndDrag	public	TCustomImageList
FieldAddress	public	TObject
FileLoad	public	TCustomImageList
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetBitmap	public	TCustomImageList
GetHotSpot	public	TCustomImageList
GetIcon	public	TCustomImageList
GetImageBitmap	public	TCustomImageList
GetInstRes	public	TCustomImageList
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetMaskBitmap	public	TCustomImageList
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetResource	public	TCustomImageList
HandleAllocated	public	TCustomImageList
HasParent	public	TComponent
HideDragImage	public	TCustomImageList
InheritsFrom	public	TObject
InitInstance	public	TObject
Insert	public	TCustomImageList
InsertComponent	public	TComponent
InsertIcon	public	TCustomImageList
InsertMasked	public	TCustomImageList
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
Move	public	TCustomImageList
NewInstance	public	TObject
Overlay	public	TCustomImageList
RegisterChanges	public	TCustomImageList
RemoveComponent	public	TComponent
Replace	public	TCustomImageList

Method	Visibility	Documented in
ReplaceIcon	public	TCustomImageList
ReplaceMasked	public	TCustomImageList
ResInstLoad	public	TCustomImageList
ResourceLoad	public	TCustomImageList
SafeCallException	public	TComponent
SetDragImage	public	TCustomImageList
ShowDragImage	public	TCustomImageList
UnRegisterChanges	public	TCustomImageList

TInplaceEdit

Unit: Grids

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomEdit • TCustomMaskEdit • TInplaceEdit

TInplaceEdit provides text editing capabilities for grid controls.

Description *TInplaceEdit* is a helper object tailored to work with grid controls. It provides a multiline editing window that appears in the appropriate cell of a grid control. *TInplaceEdit* provides services so that the grid can influence the response of the control to user actions, and can move or resize the editing window as cells are moved or resized. The *TInplaceEdit* window has no border, so it does not stand out as a separate control. *TInplaceEdit* is not a data-aware control, but can work with a data-aware grid such as *TDBGrid*, allowing the grid to fetch or post the text, and allowing the grid to impose a mask to limit the text of the edit control to data that is valid.

See also *TCustomGrid* object, *TDBEdit* object, *TDBGrid* object

Properties

The following table lists the properties of *TInplaceEdit*

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent

Property	Visibility	Default	Documented in
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
EditText	public		TCustomMaskEdit
Enabled	public	True	TControl
Grid	protected, read-only		TInplaceEdit
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
IsMasked	public, read-only		TCustomMaskEdit
Left	published		TControl
Modified	public		TCustomEdit
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentWindow	public		TWinControl
SelLength	public		TCustomEdit
SelStart	public		TCustomEdit
SelText	public		TCustomEdit
ShowHint	public		TControl
Showing	public, read-only		TWinControl
TabOrder	public	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent
Text	public		TCustomMaskEdit
Top	published		TControl
VCLComObject	public		TComponent
Visible	public	True	TControl
Width	published		TControl
WindowProc	public		TControl

Grid

Grid specifies the grid object that owns and controls this *TInplaceEdit*.

property Grid: TCustomGrid;

Description Read *Grid* to obtain the grid object for which *TInplaceEdit* provides editing services. *TInplaceEdit* uses the *Grid* property so that it can receive information about the text to be edited and so that it can pass on relevant events.

See also *TCustomGrid* object, *UpdateContents* method, *UpdateLoc* method

Methods

The following table lists the methods of *TInplaceEdit*

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BoundsChanged	protected	TInplaceEdit
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TCustomMaskEdit
ClearSelection	public	TCustomEdit
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
CopyToClipboard	public	TCustomEdit
Create	public	TInplaceEdit
CreateParams	protected	TInplaceEdit
CreateParented	public	TWinControl
CutToClipboard	public	TCustomEdit
DblClick	protected	TInplaceEdit
DefaultHandler	public	TObject
Deselect	public	TInplaceEdit
Destroy	public	TWinControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EditCanModify	protected	TInplaceEdit

Method	Visibility	Documented in
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetSelTextBuf	public	TCustomEdit
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TCustomMaskEdit
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TInplaceEdit
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TInplaceEdit
KeyDown	protected	TInplaceEdit
KeyPress	protected	TInplaceEdit
KeyUp	protected	TInplaceEdit
MethodAddress	public	TObject
MethodName	public	TObject
Move	public	TInplaceEdit
NewInstance	public	TObject
PaintTo	public	TWinControl
PasteFromClipboard	public	TCustomEdit
Perform	public	TControl
PosEqual	public	TInplaceEdit
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent

Method	Visibility	Documented in
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SelectAll	public	TCustomEdit
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TInplaceEdit
SetSelTextBuf	public	TCustomEdit
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateContents	protected	TInplaceEdit
UpdateControlState	public	TWinControl
UpdateLoc	public	TInplaceEdit
ValidateEdit	public	TCustomMaskEdit
WndProc	protected	TInplaceEdit

BoundsChanged

BoundsChanged sets a slight indentation in the text area of the window after the grid repositions the edit control.

```
procedure BoundsChanged; virtual;
```

Description *BoundsChanged* reinitializes the editing window after the control has been moved or resized. It creates a slight indentation in the text formatting area so that the contents of the editor do not appear to crowd the edges of the cell.

BoundsChanged is exposed as a protected method so that derived objects can add further enhancements when the editing window changes size or location.

See also *Move* method, *UpdateLoc* method, *PosEqual* method

Create

Create is the constructor for *TInplaceEdit*.

```
constructor Create(AOwner: TComponent); override;
```

Description *Create* overrides the default constructor to make sure that the control has no border to provide a visual indication that the control exists.

See also *Hide* method, *BoundsChanged* method

CreateParams

The *CreateParams* method initializes a window-creation parameter record passed in the *Params* parameter.

```
type
  TCreateParams = record
    Caption: PChar;
    Style: Longint;
    ExStyle: Longint;
    X, Y: Integer;
    Width, Height: Integer;
    WndParent: HWND;
    Param: Pointer
    WindowClass: TWndClass;
    WinClassName: array[0..63] of Char;
  end;

procedure CreateParams(var Params: TCreateParams); override;
```

Description The *CreateWnd* method calls *CreateParams* to initialize the parameters it passes to *CreateWindowHandle*. *TInplaceEdit* overrides the inherited method to specify a multiline edit control.

See also *CreateWnd* method, *CreateWindowHandle* method

DblClick

DblClick informs the grid when the user double-clicks in the edit window.

```
procedure DblClick; override;
```

Description *DblClick* is called when the user double-clicks in the edit window. *DblClick* simply passes the event on to the grid. The grid must interpret and respond to the double-click. If the edit window needs to respond to the double-click, the grid must trigger the response by calling a method or setting a property of the *TInplaceEdit*.

See also *Grid* property

Deselect

Deselect removes the selection from the edit window.

```
procedure Deselect;
```

Description Use *Deselect* to cause all text in the edit window to be unselected. *Deselect* simply changes the selection, it does not change the actual text.

See also *ClearSelection* method, *SelectAll* method, *KeyPress* method

EditCanModify

EditCanModify determines whether the user can edit the *Text*.

```
function EditCanModify: Boolean; override;
```

Description *EditCanModify* is called automatically when the user tries to edit the value of *Text*. *EditCanModify* calls the *CanEditModify* method of the grid to determine whether the current cell of the grid should be edited. When *EditCanModify* returns *True*, the user is permitted to edit the text. When *EditCanModify* returns *False*, the user is prevented from editing the text.

See also *Grid* property, *TCustomGrid.CanEditModify* method, *Text* property

Hide

Hide hides the edit control window and moves input focus to the grid.

```
procedure Hide;
```

Description *Hide* makes the edit control window invisible without destroying it. *Hide* also moves the input focus from the editor window to the grid, so that the invisible window does not swallow keyboard events.

The grid calls *Hide* to hide the edit window when changing out of edit mode. It also calls *Hide* when scrolling the cell that is being edited out of the grid window.

See also *Grid* property

Invalidate

Invalidate invalidates the edit window and the corresponding portion of the grid window.

```
procedure Invalidate;
```

Description *Invalidate* ensures that the portion of the grid window under the edit control is invalidated at the same time as the edit window. This is important when the edit window is hidden or destroyed so that the grid window can repaint the cell that was covered by the edit window.

See also *Grid* property, *Hide* method

KeyDown

KeyDown overrides the inherited method to pass many keystrokes on to the grid.

```
type TShiftState = set of (ssShift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle, ssDouble);
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

Description When the user presses a key, *KeyDown* checks whether the *KeyDown* event should be processed by the edit control or the grid. The *Enter* key is always passed on to the grid. The *F2* key calls *Deselect* to remove the current selection. Navigation keystrokes are passed on to the grid when the movement cannot be accommodated within the text window. For example, if the user presses the left arrow key when the cursor is already at the left edge of the edit window, the keystroke is passed to the grid which moves focus to the previous cell. If the cursor is in the middle of a line, the keystroke is processed by the edit window, which moves left a single character.

KeyDown calls the *OnKeyDown* event handler for the grid, if there is one, before calling the inherited *KeyDown* for the edit control.

See also *Grid* property, *KeyPress* method, *KeyUp* method, *TCustomMaskEdit.KeyDown* method

KeyPress

KeyPress overrides the inherited method to give the grid first chance at handling the keystroke.

```
procedure KeyPress(var Key: Char); override;
```

Description *KeyPress* allows the grid to handle the keystroke, and only processes keystrokes that the grid does not fully handle. If the grid is not in edit mode after handling the keystroke, *KeyPress* throws away the keystroke. Keystroke processes the *Return* Key and calls the inherited method to handle all other unprocessed keystrokes.

Pressing the *Return* key causes *KeyPress* to select all the text, unless it is already selected, in which case it causes *KeyPress* to deselect all the text.

See also *Grid* property, *SelectAll* method, *Deselect* method, *TCustomMaskEdit.KeyPress* method, *KeyDown* method, *KeyUp* method

KeyUp

KeyUp overrides the inherited method to pass all *KeyUp* events to the grid.

```
type TShiftState = set of (ssShift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle, ssDouble);

procedure KeyUp(var Key: Word; Shift: TShiftState); override;
```

Description The inplace edit control does not process the *KeyUp* event at all. It simply passes this event on to the grid, which may or may not process the event.

See also *Grid* property, *KeyDown* method, *KeyPress* method

Move

Move changes the position of the edit window and repaints the contents.

```
procedure Move(const Loc: TRect);
```

Description The grid calls *Move* to move the edit window to the indicated rectangle when the corresponding cell moves or resizes. The position of *Loc* is relative to the grid's client area.

See also *Grid* property, *UpdateLoc* method

PosEqual

PosEqual indicates whether the edit window has the size and position indicated by *Rect*.

```
function PosEqual(const Rect: TRect): Boolean;
```

Description Use *PosEqual* to determine if the edit window occupies a particular rectangle, where *Rect* is given in global coordinates. *PosEqual* can be used to determine whether the inplace edit control represents a particular cell in the grid, by checking the bounding rectangle of the cell.

See also *BoundsRect* property, *Grid* object

SetFocus

SetFocus gives the editor window input focus if it is visible.

```
procedure SetFocus;
```

Description Use *SetFocus* to cause keyboard events to go to the editor window. Focus will not move to the window if it is hidden by a call to the *Hide* method.

See also *KeyPress* method, *KeyDown* method, *KeyUp* method, *Hide* method

UpdateContents

UpdateContents sets the *EditMask* and *Text* to the values specified by the grid.

```
procedure UpdateContents; virtual;
```

Description *UpdateContents* calls the *GetEditMask* and *GetEditText* methods of the grid, and sets the *EditMask* and *Text* properties accordingly. *UpdateContents* allows the grid to bring up the inplace edit control with the contents of a particular cell, and to impose a mask to restrict the editor to valid input for the particular cell.

See also *Grid* property, *EditMask* property, *Text* property, *TCustomGrid.GetEditMask* method, *TCustomGrid.GetEditText* method

UpdateLoc

UpdateLoc changes the position of the edit window but does not repaint the contents.

```
procedure UpdateLoc(const Loc: TRect);
```

Description The grid calls *UpdateLoc* to move the edit window when the grid is moved. Unlike the *Move* method, *UpdateLoc* does not invalidate the edit window to cause a repaint.

See also *Grid* property, *Move* method, *Invalidate* method

WndProc

WndProc supplements the inherited method to add special processing to some Windows messages.

```
procedure WndProc(var Message: TMessage); override;
```

Description *WndProc* is the first method that receives messages for a Delphi control.

WndProc for *TInplaceEdit* passes focus events on to the grid. It calls the inherited method for all other Windows messages. *WndProc* converts mouse clicks to double clicks when the first click was received by the grid within the double-click time period.

See also *Grid* property, *TWinControl.WndProc* method

TIntegerField

Unit: DB

TObject • TPersistent • TComponent • TField • TNumericField • TIntegerField

A *TIntegerField* object represents a signed 32-bit integer field in a dataset.

Description Integer fields can hold values in the range –2,147,483,648 to 2,147,483,647, and are frequently used to hold values that are large, signed whole numbers.

TIntegerField introduces new properties to convert between integer values and other data types, and to limit the range of valid integers that the field will permit. As a descendant of *TNumericField*, *TIntegerField* includes many properties, methods, and events that are useful for managing the value and properties of a numeric field in a database.

TIntegerField is the direct ancestor of three other field types that represent whole numbers: *TAutoIncField*, *TSmallintField*, and *TWordField*.

See also *TAutoIncField* object, *TDataSet* object, *TSmallintField* object, *TWordField* object

Properties

The following table lists the properties of *TIntegerField*:

Property	Visibility	Default	Documented in
Alignment	published	taRightJustify	TField
AsBoolean	public		TField
AsCurrency	public		TField
AsDateTime	public		TField
AsFloat	public		TIntegerField
AsInteger	public		TIntegerField
AsString	public		TIntegerField
AsVariant	public		TIntegerField
AttributeSet	public		TField
Calculated	public	False	TField
CanModify	public, read-only		TField
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ConstraintErrorMessage	published		TField
CurValue	public, read-only		TField
CustomConstraint	published		TField
DataSet	public		TField
DataSize	public, read-only		TIntegerField
DataType	public, read-only		TField
DefaultExpression	published		TField
DesignInfo	public		TComponent
DisplayFormat	published		TNumericField
DisplayLabel	published		TField
DisplayName	public, read-only		TField
DisplayText	public, read-only		TField
DisplayWidth	published		TField
EditFormat	published		TNumericField
EditMask	public		TField
EditMaskPtr	public, read-only		TField
FieldKind	published	fkData	TField
FieldName	published		TField
FieldNo	public, read-only		TField
ImportedConstraint	published		TField
Index	published		TField
IsIndexField	public, read-only		TField
IsNull	public, read-only		TField
KeyFields	published		TField
Lookup	public		TField

Property	Visibility	Default	Documented in
LookupCache	published	False	TField
LookupDataSet	published		TField
LookupKeyFields	published		TField
LookupList	public, read-only		TField
LookupResultField	published		TField
MaxValue	published	0	TIntegerField
MinValue	published	0	TIntegerField
Name	published		TComponent
NewValue	public		TField
Offset	public, read-only		TField
OldValue	public, read-only		TField
Origin	published		TField
Owner	public, read-only		TComponent
ReadOnly	published	False	TField
Required	published	False	TField
Size	public		TField
Tag	published	0	TComponent
Text	public		TField
ValidChars	public		TField
Value	public		TIntegerField
VCLComObject	public		TComponent
Visible	published	True	TField

AsFloat

AsFloat converts the value of the integer field to or from a floating-point value.

property AsFloat: Double;

Description Use *AsFloat* to

- Assign the integer field value to a floating-point variable.
- Round a floating-point value to an integer value and assign it to an integer field.

For example, this code assigns a floating-point value to an integer field value, rounding the value to 12:

```
MyIntegerField.AsFloat := 12.23;
```

To assign an integer field value to a floating-point value, use code similar to this:

```
FloatVariable := MyIntegerField.AsFloat;
```

If the value assigned to the field does not fall within the range specified by the *MinValue* and *MaxValue* properties, an *EDatabaseError* exception is raised.

See also *MaxValue property, MinValue property*

AsInteger

AsInteger gets or sets the value of an integer field as a 32-bit integer.

property AsInteger: Longint;

Description Use *AsInteger* to work with the value of an integer field as a 32-bit integer. Integer field components store and manipulate their values as 32-bit integers, so *AsInteger* is the same as the *Value* property.

Long integer values need no conversion, because the native storage format of an integer field is a 32-bit integer. Descendants of *TIntegerField* that represent other types of integer fields convert the value of the physical database field to a 32-bit integer for the *AsInteger* property.

If the value assigned to the field does not fall within the range specified by the *MinValue* and *MaxValue* properties, an *EDatabaseError* exception is raised.

See also *DataType* property, *MaxValue property, MinValue property*

AsString

AsString converts the value of an integer field to or from a string value.

property AsString: **string**;

Description Use *AsString* to

- Assign the integer field value to string variable.
- Assign a string value to the integer field.

For example, this code assigns a string value to an integer field value:

```
MyIntegerField.AsString := '27';
```

To assign an integer field value to a string, use code similar to this:

```
NewString := MyIntegerField.AsString;
```

If the string assigned to the field does not represent a valid integer value, an *EDatabaseError* exception is raised. Also, if the assigned string does not convert to an integer value that falls within the range specified by the *MinValue* and *MaxValue* properties, an *EDatabaseError* exception is raised.

See also *MaxValue property, MinValue property*

AsVariant

AsVariant converts the value of an integer field to or from a variant value.

property AsVariant: Variant;

Description Use *AsVariant* to

- Assign the integer field value to variant variable.
- Assign a variant value to the integer field.

Using variants is useful when you don't know the actual field type and don't want to use one of the other *As...* properties to convert the field value to another type.

If the value assigned to the field can't be converted to a valid integer or does not fall within the range specified by the *MinValue* and *MaxValue* properties, an *EDatabaseError* exception is raised.

See also *AsFloat* property, *AsInteger* property, *AsString* property, *EDatabaseError* object, *MaxValue* property, *MinValue* property

DataSize

DataSize is the amount of memory that is needed to store the value of an integer field in its native database format.

property DataSize: Word;

Description Check *DataSize* to determine the number of bytes required to store an integer field component's value. Use *DataSize* to determine the size needed for a buffer when working with the *GetData* and *SetData* methods.

For integer fields, *DataSize* is always 4. Some descendants of *TIntegerField* such as *TSmallintField*, represent database fields with smaller values of *DataSize*.

See also *GetData* method, *SetData* method, *TSmallintField* object, *Value* property

MaxValue

MaxValue limits the maximum value in the integer field.

property MaxValue: Longint;

Description Use *MaxValue* to get or set the maximum value that can be entered into the field. If a value greater than *MaxValue* is entered, an *EDatabaseError* exception is raised.

If the value of *MaxValue* is 0, the integer field can have a value up to the maximum value that can be represented by the native database type. For 32-bit integer fields, this value is 2147483647.

See also *MinValue* property

MinValue

MinValue limits the minimum value in the integer field.

property MaxValue: Longint;

Description Use *MinValue* to get or set the minimum value that can be entered into the field. If a value smaller than *MinValue* is entered, an *EDatabaseError* exception is raised.

If the value of MinValue is 0, the integer field can have a value as small as the native database type can represent. For 32-bit integer fields, this minimum value is –2147483648.

See also *MaxValue* property

Value

Value is the actual data in the integer field.

property Value: Longint;

Description Use *Value* to read data directly from and write data directly to an integer field. Using the *Value* property is the same as using the *AsInteger* property. Use *Value* when you know the field component is a 32-bit integer field. Use *AsInteger* when working with a generic field component.

See also *AsInteger* property

Events

The following table lists the events of *TIntegerField*:

Event	Visibility	Documented in
OnChange	published	TField
OnGetText	published	TField
OnSetText	published	TField
OnValidate	published	TField

Methods

The following table lists the methods of *TIntegerField*:

Method	Visibility	Documented in
Assign	public	TField
AssignValue	public	TField
ClassInfo	public	TObject
ClassName	public	TObject

Method	Visibility	Documented in
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TField
Create	public	TIntegerField
DefaultHandler	public	TObject
Destroy	public	TField
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
FocusControl	public	TField
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetData	public	TField
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
IsBlob	public	TField
IsValidChar	public	TField
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RefreshLookupList	public	TField
RemoveComponent	public	TComponent
SafeCallException	public	TComponent
SetData	public	TField
SetFieldType	public	TField
Validate	public	TField

Create

Create creates an instance of *TIntegerField*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to create and initialize an instance of *TIntegerField*. After calling the inherited constructor, *Create* sets the *DataType* to *ftInteger*, the *MinValue* to –2147483648, and the *MaxValue* to 2147483647.

It is seldom necessary to call *Create* directly, because an integer field component is instantiated automatically for all integer fields in a dataset.

See also *DataType* property, *MaxValue property*, *MinValue property*, *TField.Create* method

TISAPIApplication

Unit: ISAPIApp

TObject • TPersistent • TComponent • TWebApplication • TISAPIApplication

TISAPIApplication encapsulates an ISAPI or NSAPI server application.

Description *TISAPIApplication* provides the fundamental behavior of an ISAPI or NSAPI server application. The HTTP server calls the methods of *TISAPIApplication* when it receives an HTTP request message. *TISAPIApplication* creates a thread to manage the request message, and a creates a new instance of the Web module for that thread, or reuses an old instance that was deactivated after a previous response was sent.

TISAPIApplication creates *TISAPIRequest* and *TISAPIResponse* objects to represent the request message and any response that should be sent in return, passes these to the dispatcher so that the response can be filled in, and sends the response if it was not already sent by a *TWebActionItem*.

Each ISAPI server project automatically declares an *Application* variable of type *TISAPIApplication* as the instance of the application. This declaration overrides the *Application* variable declared in the Forms unit if the ISAPIApp unit is declared last in the uses list of the project file. Every ISAPI server application must include the ISAPIApp unit.

To work with a CGI server, use the *TCGIApplication* object instead.

See also *TCGIApplication* object, *TISAPIRequest* object, *TISAPIResponse* object, *TWebActionItem* object, *TWebDispatcher* object

Properties

The following table lists the properties of *TISAPIApplication*:

Property	Visibility	Default	Documented in
ActiveCount	public, read-only		TWebApplication
CacheConnections	public		TWebApplication
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DesignInfo	public		TComponent
InactiveCount	public, read-only		TWebApplication
MaxConnections	public		TWebApplication
Name	published		TComponent
Owner	public, read-only		TComponent
Tag	published	0	TComponent
Title	public		TWebApplication
VCLComObject	public		TComponent

Methods

The following table lists the methods of *TISAPIApplication*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TWebApplication
CreateForm	public	TWebApplication
DefaultHandler	public	TObject
Destroy	public	TWebApplication
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject

Method	Visibility	Documented in
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetExtensionVersion	public	TISAPIApplication
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
HttpExtensionProc	public	TISAPIApplication
InheritsFrom	public	TObject
Initialize	public	TWebApplication
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
Run	public	TWebApplication
SafeCallException	public	TComponent
TerminateExtension	public	TISAPIApplication

GetExtensionVersion

GetExtensionVersion returns the application name and version number.

```
type THSE_VERSION_INFO = packed record
  dwExtensionVersion: DWORD;
  lpszExtensionDesc: array [0..HSE_MAX_EXT_DLL_NAME_LEN-1] of Char;
end;

function GetExtensionVersion(var Ver: THSE_VERSION_INFO): BOOL;
```

Description The HTTP server calls *GetExtensionVersion* to obtain the application name and version number. *GetExtensionVersion* returns the version number for the HTTP functions in the runtime library, and the *Title* property of the ISAPI application object.

See also *Title* property

HttpExtensionProc

HttpExtensionProc responds to an HTTP request message.

```
type TEXTENSION_CONTROL_BLOCK = packed record
  cbSize: DWORD;                    // size of this struct.
  dwVersion: DWORD;                 // version info of this spec
  ConnID: HCONN;                    // Context number not to be modified!
  dwHttpStatusCode: DWORD;          // HTTP Status code
  lpszLogData: array [0..HSE_LOG_BUFFER_LEN-1] of Char; // log info for this DLL
  lpszMethod: PChar;                // REQUEST_METHOD
  lpszQueryString: PChar;           // QUERY_STRING
  lpszPathInfo: PChar;              // PATH_INFO
  lpszPathTranslated: PChar;        // PATH_TRANSLATED
  cbTotalBytes: DWORD;              // Total bytes indicated from client
  cbAvailable: DWORD;               // Available number of bytes
  lpbData: Pointer;                 // pointer to cbAvailable bytes
  lpszContentType: PChar;           // Content type of client data
  GetServerVariable: TGetServerVariableProc;
  WriteClient: TWriteClientProc;
  ReadClient: TReadClientProc;
  ServerSupportFunction: TServerSupportFunctionProc;
end;

function HttpExtensionProc(var ECB: TEXTENSION_CONTROL_BLOCK): DWORD;
```

Description The HTTP server calls *HttpExtensionProc* when it receives an HTTP request message. *HttpExtensionProc* creates the *TISAPIRequest* and *TISAPIResponse* objects for the HTTP request message, and calls the *HandleRequest* message so that the response can be filled out and sent. Finally, *HttpExtensionProc* frees the request and response objects.

If the response is filled out and sent, *HttpExtensionProc* returns HSE_STATUS_SUCCESS. Otherwise it returns HSE_STATUS_ERROR.

See also *TISAPIRequest* object, *TISAPIResponse* object, *TWebApplication.HandleRequest* method, *TWebDispatcher* object

TerminateExtension

TerminateExtension returns *True*.

```
function TerminateExtension(dwFlags: DWORD): BOOL;
```

Description The HTTP server calls *TerminateExtension* when it wants to shut down the ISAPI application. *TerminateExtension* always returns *True*, allowing the server to shut down.

TLabel

Unit: StdCtrls

Palette: Standard

TObject • TPersistent • TComponent • TControl • TGraphicControl • TCustomLabel • TLabel

TLabel is a nonwindowed control that displays text on a form.

Description

Use *TLabel* to add text to a form. This text can be used to label anther control, and can set focus to that control when the user types an accelerator key.

TLabel is not a descendant of *TWinControl*, so it does not have its own window and can't receive direct input from the keyboard. To add an object to a form that can respond to keyboard input (other than setting focus to another object when an accelerator key is pressed) in addition to displaying text, use *TStaticText*.

To add an object to a form that displays text that a user can scroll or edit, use *TEdit*.

See also

TEdit object, *TStaticText* object

Properties

The following table lists the properties of *TLabel*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
Alignment	published	taLeftJustify	TCustomLabel
AutoSize	published	True	TCustomLabel
BoundsRect	public		TControl
Canvas	public, read-only		TGraphicControl
Caption	published		TControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TControl
ClientRect	public, read-only		TControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl

Property	Visibility	Default	Documented in
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
FocusControl	published		TCustomLabel
Font	published		TControl
Height	published		TControl
Hint	published		TControl
Layout	published	tlTop	TCustomLabel
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	True	TControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
PopupMenu	published		TControl
ShowAccelChar	published	True	TCustomLabel
ShowHint	published		TControl
Tag	published	0	TComponent
Top	published		TControl
Transparent	published	False	TCustomLabel
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl
WordWrap	published	False	TCustomLabel

Events

The following table lists the events of *TLabel*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl

Event	Visibility	Documented in
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the events of *TLabel*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
Create	public	TCustomLabel
DefaultHandler	public	TObject
Destroy	public	TGraphicControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTextBuf	public	TControl
GetTextLen	public	TControl
HasParent	public	TComponent

Method	Visibility	Documented in
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
Invalidate	public	TControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Perform	public	TControl
Refresh	public	TControl
RemoveComponent	public	TComponent
Repaint	public	TControl
SafeCallException	public	TComponent
ScreenToClient	public	TControl
SendToBack	public	TControl
SetBounds	public	TControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TControl

TListBox

Unit: StdCtrls

Palette: Standard

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomListBox • TListBox

TListBox is a wrapper for a Windows list box control.

Description

TListBox implements the generic behavior introduced in *TCustomListbox*. *TListBox* publishes many of the properties inherited from *TCustomListBox*, but does not introduce any new behavior.

Use *TListBox* to display a scrollable list of items that users can select, add or delete. For specialized list boxes use other descendant classes of *TCustomListBox* or derive from it.

See also

TComboBox object, *TListView* object

Properties

The following table lists the properties of *TListBox*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
BorderStyle	published	bsSingle	TCustomListBox
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Canvas	public, read-only		TCustomListBox
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
Columns	published	0	TCustomListBox
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
ExtendedSelect	published	True	TCustomListBox
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
ImeMode	published	imDontCare	TWinControl
ImeName	published		TWinControl
IntegralHeight	published	False	TCustomListBox
ItemHeight	published		TCustomListBox
ItemIndex	public		TCustomListBox
Items	published		TCustomListBox
Left	published		TControl
MultiSelect	published	False	TCustomListBox
Name	published		TControl

Property	Visibility	Default	Documented in
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
SelCount	public, read-only		TCustomListBox
Selected	public		TCustomListBox
ShowHint	published		TControl
Showing	public, read-only		TWinControl
Sorted	published	False	TCustomListBox
Style	published	lbStandard	TCustomListBox
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
TabWidth	published	0	TCustomListBox
Tag	published	0	TComponent
Top	published		TControl
TopIndex	public		TCustomListBox
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Events

The following table lists the events of *TListBox*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnDrawItem	published	TCustomListBox
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl

Event	Visibility	Documented in
OnMeasureItem	published	TCustomListBox
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TListBox:*

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TCustomListBox
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomListBox
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TCustomListBox
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl

Method	Visibility	Documented in
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
ItemAtPos	public	TCustomListBox
ItemRect	public	TCustomListBox
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl

Method	Visibility	Documented in
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

TListView

Unit: ComCtrls

Palette: Win32

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomListView • TListView

TListView displays a list of items in a variety of ways.

Description Use *TListView* to manage and display a list of items in a form. The items can be displayed in columns with column headers and sub-items, or vertically or horizontally, with small or large icons.

TListView publishes many of the properties, events, and methods of *TCustomListView*, but does not introduce any new behavior.

See also *TListColumn* object, *TListItem* object

Properties

The following table lists the properties of *TListView*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
AllocBy	published	0	TCustomListView
BorderStyle	published	bsSingle	TCustomListView
BoundingRect	public, read-only		TCustomListView
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Checkboxes	published	False	TCustomListView
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
Column	public, read-only		TCustomListView
ColumnClick	published	True	TCustomListView
Columns	published		TCustomListView
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent

Property	Visibility	Default	Documented in
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
DropTarget	public		TCustomListView
Enabled	published	True	TControl
Font	published		TControl
GridLines	published	False	TCustomListView
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
HideSelection	published	True	TCustomListView
Hint	published		TControl
HotTrack	published	False	TCustomListView
IconOptions	published		TCustomListView
ItemFocused	public		TCustomListView
Items	published		TCustomListView
LargeImages	published		TCustomListView
Left	published		TControl
MultiSelect	published	False	TCustomListView
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	False	TControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ReadOnly	published	False	TCustomListView
RowSelect	published	False	TCustomListView
SelCount	public, read-only		TCustomListView
Selected	public		TCustomListView
ShowColumnHeaders	published	True	TCustomListView
ShowHint	published		TControl

Property	Visibility	Default	Documented in
Showing	public, read-only		TWinControl
SmallImages	published		TCustomListView
SortType	published	stNone	TCustomListView
StateImages	published		TCustomListView
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent
Top	published		TControl
TopItem	public, read-only		TCustomListView
VCLComObject	public		TComponent
ViewOrigin	public, read-only		TCustomListView
ViewStyle	published	vsIcon	TCustomListView
Visible	published	True	TControl
VisibleRowCount	public, read-only		TCustomListView
Width	published		TControl
WindowProc	public		TControl

Events

The following table lists the events of *TListView*:

Event	Visibility	Documented in
OnChange	published	TCustomListView
OnChanging	published	TCustomListView
OnClick	published	TControl
OnColumnClick	published	TCustomListView
OnCompare	published	TCustomListView
OnDblClick	published	TControl
OnDeletion	published	TCustomListView
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEdited	published	TCustomListView
OnEditing	published	TCustomListView
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnInsert	published	TCustomListView
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl

Event	Visibility	Documented in
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TListView*:

Method	Visibility	Documented in
AlphaSort	public	TCustomListView
Arrange	public	TCustomListView
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomListView
CreateParented	public	TWinControl
CustomSort	public	TCustomListView
DefaultHandler	public	TObject
Destroy	public	TCustomListView
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindCaption	public	TCustomListView
FindComponent	public	TComponent

Method	Visibility	Documented in
FindData	public	TCustomListView
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetItemAt	public	TCustomListView
GetNamePath	public	TPersistent
GetNearestItem	public	TCustomListView
GetNextItem	public	TCustomListView
GetParentComponent	public	TComponent
GetSearchString	public	TCustomListView
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
IsEditing	public	TCustomListView
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl

Method	Visibility	Documented in
Scroll	public	TCustomListView
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
StringWidth	public	TCustomListView
Update	public	TWinControl
UpdateControlState	public	TWinControl
UpdateItems	public	TCustomListView

TLoginDialog

Unit: DBLogDlg

TObject • TPersistent • TComponent • TControl • TWinControl • TScrollingWinControl • TCustomForm • TForm • TLoginDialog

TLoginDialog is a login dialog box that can be used to provide login security in an application.

Description Use *TLoginDialog* to provide login security in an application. Usually a *TLoginDialog* object is created with the *LoginDialogEx* function and is used to provide secure access to local database tables. However, *TLoginDialog* can also be used as a base object for other types of login dialog boxes.

See also *LoginDialogEx* function, *TPasswordDialog* object

Properties

The following table lists the properties of *TLoginDialog*:

Property	Visibility	Default	Documented in
Active	public, read-only		TCustomForm
ActiveControl	published		TCustomForm
ActiveMDIChild	public, read-only		TCustomForm
ActiveOleControl	public		TCustomForm
Align	public	alNone	TControl
AutoScroll	published	True	TScrollingWinControl
BorderIcons	published	[biSystemMenu, biMinimize, biMaximize]	TCustomForm
BorderStyle	published	bsSizeable	TCustomForm
BoundsRect	public		TControl

Property	Visibility	Default	Documented in
Brush	public, read-only		TWinControl
Canvas	public, read-only		TCustomForm
Caption	published		TControl
ClientHandle	public, read-only		TCustomForm
ClientHeight	published		TCustomForm
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TCustomForm
ClientWidth	published		TCustomForm
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published	True	TWinControl
Cursor	published	crDefault	TControl
Designer	public		TCustomForm
DesignInfo	public		TComponent
DropTarget	public		TCustomForm
Enabled	published	True	TControl
Font	published		TControl
FormStyle	published	fsNormal	TCustomForm
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
HelpFile	published		TCustomForm
Hint	published		TControl
HorzScrollBar	published		TScrollingWinControl
Icon	published		TCustomForm
KeyPreview	published	False	TCustomForm
Left	published		TControl
MDIChildCount	public, read-only		TCustomForm
MDIChildren	public, read-only		TCustomForm
Menu	published		TCustomForm
ModalResult	public		TCustomForm
Name	published		TControl
ObjectMenuItem	published		TCustomForm

Property	Visibility	Default	Documented in
OleFormObject	public		TCustomForm
Owner	public, read-only		TComponent
Parent	public		TCustomForm
ParentFont	published	False	TControl
ParentWindow	public		TWinControl
PixelsPerInch	published		TCustomForm
PopupMenu	published		TControl
Position	published	poDesigned	TCustomForm
PrintScale	published	poProportional	TCustomForm
Scaled	published	True	TCustomForm
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	public	–1	TWinControl
TabStop	public	False	TWinControl
Tag	published	0	TComponent
TileMode	public	tbHorizontal	TCustomForm
Top	published		TControl
VCLComObject	public		TComponent
VertScrollBar	published		TScrollingWinControl
Visible	published	False	TCustomForm
Width	published		TControl
WindowMenu	published		TCustomForm
WindowProc	public		TControl
WindowState	published	wsNormal	TCustomForm

L

Events

The following table lists the events of *TLoginDialog*:

Event	Visibility	Documented in
OnActivate	published	TCustomForm
OnClick	published	TControl
OnClose	published	TCustomForm
OnCloseQuery	published	TCustomForm
OnCreate	published	TCustomForm
OnDblClick	published	TControl
OnDeactivate	published	TCustomForm
OnDestroy	published	TCustomForm
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnHelp	published	TCustomForm

Event	Visibility	Documented in
OnHide	published	TCustomForm
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnPaint	published	TCustomForm
OnResize	published	TCustomForm
OnShow	published	TCustomForm

Methods

The following table lists the methods of *TLoginDialog*:

Method	Visibility	Documented in
ArrangeIcons	public	TForm
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
Cascade	public	TForm
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
Close	public	TCustomForm
CloseQuery	public	TCustomForm
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomForm
CreateNew	public	TCustomForm
CreateParented	public	TWinControl
DefaultHandler	public	TObject
DefocusControl	public	TCustomForm
Destroy	public	TCustomForm
DestroyComponents	public	TComponent

Method	Visibility	Documented in
Destroying	public	TComponent
DisableAlign	public	TWinControl
DisableAutoRange	public	TScrollingWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EnableAutoRange	public	TScrollingWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
FocusControl	public	TCustomForm
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetFormImage	public	TCustomForm
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TCustomForm
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Next	public	TForm
PaintTo	public	TWinControl
Perform	public	TControl

Method	Visibility	Documented in
Previous	public	TForm
Print	public	TCustomForm
Realign	public	TWinControl
Refresh	public	TControl
Release	public	TCustomForm
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
ScrollInView	public	TScrollingWinControl
SendCancelMode	public	TCustomForm
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TCustomForm
SetFocusedControl	public	TCustomForm
SetTextBuf	public	TControl
Show	public	TCustomForm
ShowModal	public	TCustomForm
Tile	public	TForm
Update	public	TWinControl
UpdateControlState	public	TWinControl

TMainMenu

Unit: Menus | Palette: Standard

TObject • TPersistent • TComponent • TMenu • TMainMenu

TMainMenu encapsulates a menu bar and its accompanying drop-down menus for a form.

Description Use *TMainMenu* to provide the main menu for a form. To begin designing a menu, add a main menu to a form, and double-click the component.

TMainMenu introduces properties and methods for merging the drop-down menus of the main menu with the main menu of another form, and for assisting in the menu negotiation process of an OLE container.

See also *TMenuItem* object, *TPopupMenu* object

Properties

The following table lists the properties of *TMainMenu*:

Property	Visibility	Default	Documented in
AutoMerge	published	False	TMainMenu
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DesignInfo	public		TComponent
Handle	public, read-only		TMainMenu
Items	published, read-only		TMenu
Name	published		TComponent
Owner	public, read-only		TComponent
Tag	published	0	TComponent
VCLComObject	public		TComponent
WindowHandle	public		TMenu

AutoMerge

AutoMerge determines if the main menus of forms other than the main form merge with the main menu of the main form in non-MDI applications at runtime.

```
property AutoMerge: Boolean;
```

Description To merge a form's menus with the main menu in the main form, set the *AutoMerge* property of each main menu you want merged to *True*. Make sure that the *AutoMerge* property of the main menu of the main form (the one other menus are merged into) remains *False*. How menus merge depends on the value of the *GroupIndex* property for each menu item.

If the application is an MDI application (the *FormStyle* properties are set so the main form is a parent form and subsequent forms are child forms), menu merging occurs automatically without using the *AutoMerge* property.

See also *FormStyle* property, *GroupIndex* property, *Merge* method

Handle

Handle provides access to the menu's window handle.

```
property Handle: HMENU;
```

Description Use the *Handle* property to call a Windows API function that requires a Windows menu handle.

For example, the following line uses the Windows API function *HiliteMenuItem* to highlight the first menu item in *MainMenu1* on *Form1*.

```
HiliteMenuItem(Form1.Handle, MainMenu1.Handle, 0, MF_BYPOSITION+MF_HILITE);
```

If the menu has an associated OLE menu handle, *Handle* is the OLE2 menu handle. Otherwise, *Handle* is the Windows menu handle for the menu that contains exactly the elements described in the *Items* property.

See also *SetOle2MenuHandle* method

Methods

The following table lists the methods of *TMainMenu*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TMenu
DefaultHandler	public	TObject
Destroy	public	TMenu
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
DispatchCommand	public	TMenu
DispatchPopup	public	TMenu
FieldAddress	public	TObject
FindComponent	public	TComponent
FindItem	public	TMenu
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetHelpContext	public	TMenu
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent

Method	Visibility	Documented in
GetOle2AcceleratorTable	public	TMainMenu
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
IsShortCut	public	TMenu
Merge	public	TMainMenu
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PopulateOle2Menu	public	TMainMenu
RemoveComponent	public	TComponent
SafeCallException	public	TComponent
SetOle2MenuHandle	public	TMainMenu
Unmerge	public	TMainMenu

GetOle2AcceleratorTable

GetOle2AcceleratorTable returns the main menu's accelerator table.

```
procedure GetOle2AcceleratorTable(var AccelTable: HAccel; var AccelCount: Integer; Groups:
  array of Integer);
```

Description Call *GetOle2AcceleratorTable* to obtain a handle to an accelerator table for all menu items with a *GroupIndex* value that appears in the *Groups* parameter. The *AccelTable* parameter is set to the accelerator table handle, and the *AccelCount* is set to the number of entries in the table. *GetOle2AccleratorTable* is used by OLE containers when merging menus.

See also *TMenuItem.GroupIndex* property, *TMenuItem.ShortCut* property

Merge

Merge combines a main menu of one form with a main menu of another in non-MDI applications.

```
procedure Merge(Menu: TMainMenu);
```

Description Call *Merge* to merge another main menu with this one. Specify the menu you want merged with this menu as the *Menu* parameter.

For example, to use the main menu of the first form in an application as the main menu for the application, call *Merge* to merge the main menu of any additional forms in the application with the main menu of the first form.

Depending on the value of the *GroupIndex* property of menu items on the main menu, the merged menu items can replace menu items on the menu bar, or insert menu items into the menu bar. See *GroupIndex* for information on how to do these things.

To merge and unmerge main menus automatically when other forms are displayed, change the value of the *AutoMerge* property to *True* on all forms whose menus should be merged in.

See also *AutoMerge* property, *TMenuItem.GroupIndex* property, *UnMerge* method

PopulateOle2Menu

PopulateOle2Menu populates an OLE menu with items from the main menu.

```
procedure PopulateOle2Menu(SharedMenu: HMenu; Groups: array of Integer; var Widths: array
  of Longint);
```

Description Call *PopulateOle2Menu* to merge the menu items with *GroupIndex* values specified by the *Groups* parameter into the menu that uses the *SharedMenu* handle. *PopulateOle2Menu* appends all menu items in the specified groups to the menu, and increments the corresponding entries in the *Widths* array for every menu item appended. *PopulateOle2Menu* is used by OLE containers when merging menus.

Note *PopulateOle2Menu* populates the menu specified by the *SharedMenu* parameter but does not set this as the OLE menu associated with the menu.

See also *GetOle2AcceleratorTable* property, *SetOle2MenuHandle* method, *TMenuItem.GroupIndex* property

SetOle2MenuHandle

SetOle2MenuHandle associates the handle for an OLE menu with the main menu.

```
procedure SetOle2MenuHandle(Handle: HMENU);
```

Description Call *SetOle2MenuHandle* to temporarily replace the menu of a form with the menu represented by the *Handle* parameter. To restore the original menu, call *SetOle2MenuHandle* with the *Handle* parameter set to 0. *SetOle2MenuHandle* allows an OLE container to use menu items provided by an OLE server.

See also *GetOle2AcceleratorTable* property, *PopuplateOle2Menu* method

Unmerge

Unmerge reverses the merging of two menus in a non-MDI application.

```
procedure Unmerge(Menu: TMainMenu);
```

Description Call *Unmerge* to remove items from another main menu that were added using the *Merge* method. The *Menu* parameter is the merged menu whose items should be removed from the main menu.

See also *AutoMerge* property, *Merge* method

TMaskEdit

Unit: Mask | Palette: Additional

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomEdit • TCustomMaskEdit • TMaskEdit

TMaskEdit implements a generic masked edit control.

Description Use a *TMaskEdit* object to put a masked edit control on your form. Masked edit controls validate the text the user enters against a mask that encodes the valid forms the text can take. The mask can also format text that is displayed to the user.

TMaskEdit implements the generic behavior introduced in *TCustomMaskEdit*. *TMaskEdit* publishes many of the properties and methods inherited from *TCustomMaskEdit*, but does not introduce any new behavior.

See also *TDBEdit object*, *TEdit object*

M

Properties

The following table lists the properties of *TMaskEdit*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
AutoSelect	published	True	TCustomEdit
AutoSize	published	True	TCustomEdit
BorderStyle	published	bsSingle	TCustomEdit
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
CharCase	published	ecNormal	TCustomEdit
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent

Property	Visibility	Default	Documented in
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
EditMask	published		TCustomMaskEdit
EditText	public		TCustomMaskEdit
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
ImeMode	published	imDontCare	TWinControl
ImeName	published		TWinControl
IsMasked	public, read-only		TCustomMaskEdit
Left	published		TControl
MaxLength	published	0	TCustomMaskEdit
Modified	public		TCustomEdit
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PasswordChar	published	#0	TCustomEdit
PopupMenu	published		TControl
ReadOnly	published	False	TCustomEdit
SelLength	public		TCustomEdit
SelStart	public		TCustomEdit
SelText	public		TCustomEdit
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl

Property	Visibility	Default	Documented in
TabStop	published	True	TWinControl
Tag	published	0	TComponent
Text	published		TCustomMaskEdit
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Events

The following table lists the events of *TMaskEdit*:

Event	Visibility	Documented in
OnChange	published	TCustomEdit
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TMaskEdit*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject

Method	Visibility	Documented in
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TCustomMaskEdit
ClearSelection	public	TCustomEdit
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
CopyToClipboard	public	TCustomEdit
Create	public	TCustomMaskEdit
CreateParented	public	TWinControl
CutToClipboard	public	TCustomEdit
DefaultHandler	public	TObject
Destroy	public	TWinControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetSelTextBuf	public	TCustomEdit
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TCustomMaskEdit
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl

Method	Visibility	Documented in
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
PasteFromClipboard	public	TCustomEdit
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SelectAll	public	TCustomEdit
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetSelTextBuf	public	TCustomEdit
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl
ValidateEdit	public	TCustomMaskEdit

TMediaPlayer

Unit: MPlayer

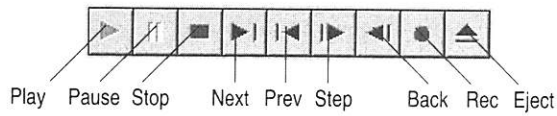
Palette: System

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomControl • TMediaPlayer

TMediaPlayer controls devices that provide a Media Control Interface (MCI) driver.

Description The *TMediaPlayer* component is a set of buttons (Play, Stop, Eject, and so on) that controls a multimedia device such as a CD-ROM drive, a MIDI sequencer, or a VCR. A multimedia device may be hardware or software.

The media player component consists of multiple buttons. These buttons can be clicked with the mouse, but are not separate objects or button components.



Button	Value	Action
Play	*btPlay*	Plays the media player
Pause	*btPause*	Pauses playing or recording. If already paused when clicked, resumes playing or recording.
Stop	*btStop*	Stops playing or recording
Next	*btNext*	Skips to the next track, or to the end if the medium doesn't use tracks
Prev	*btPrev*	Skips to the previous track, or to the beginning if the medium doesn't use tracks
Step	*btStep*	Moves forward a number of frames
Back	*btBack*	Moves backward a number of frames
Record	*btRecord*	Starts recording
Eject	*btEject*	Ejects the medium

The multimedia device is played, paused, stopped, and so on when the user clicks the corresponding button on the *TMediaPlayer* component. The device can also be controlled by the control methods that correspond to the buttons (*Play, Pause, Stop, Next, Previous, Step, Back, StartRecording,* and *Eject*).

The type of multimedia device (such as *dtWaveAudio* or *dtVideodisc*) is specified by the *DeviceType* property. If the device stores its media in a file, the name of the media file is specified by the *FileName* property. If *DeviceType* is dtAutoSelect, the media player attempts to determine the type of device from the extension of the file specified by *FileName*.

To have the media player attempt to open the device specified by *DeviceType* automatically when the media player component is created at runtime, set the *AutoOpen* property to *True*.

See also *TAnimate* object

Properties

The following table lists the properties of *TMediaPlayer*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
AutoEnable	published	True	TMediaPlayer
AutoOpen	published	False	TMediaPlayer
AutoRewind	published	True	TMediaPlayer
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Capabilities	public, read-only		TMediaPlayer
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ColoredButtons	published	[btPlay, btPause, btStop, btNext, btPrev, btStep, btBack, btRecord, btEject]	TMediaPlayer
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DeviceID	public, read-only		TMediaPlayer
DeviceType	published	dtAutoSelect	TMediaPlayer
Display	published		TMediaPlayer
DisplayRect	public		TMediaPlayer
Enabled	published	True	TControl
EnabledButtons	published	[btPlay, btPause, btStop, btNext, btPrev, btStep, btBack, btRecord, btEject]	TMediaPlayer
EndPos	public		TMediaPlayer
Error	public, read-only		TMediaPlayer
ErrorMessage	public, read-only		TMediaPlayer
FileName	published		TMediaPlayer
Frames	public		TMediaPlayer
Handle	public, read-only		TWinControl

Property	Visibility	Default	Documented in
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
Length	public, read-only		TMediaPlayer
Mode	public, read-only		TMediaPlayer
Name	published		TControl
Notify	public		TMediaPlayer
NotifyValue	public, read-only		TMediaPlayer
Owner	public, read-only		TComponent
Parent	public		TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
Position	public		TMediaPlayer
Shareable	published	False	TMediaPlayer
ShowHint	published		TControl
Showing	public, read-only		TWinControl
Start	public, read-only		TMediaPlayer
StartPos	public		TMediaPlayer
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent
TimeFormat	public		TMediaPlayer
Top	published		TControl
TrackLength	public, read-only		TMediaPlayer
TrackPosition	public, read-only		TMediaPlayer
Tracks	public, read-only		TMediaPlayer
VCLComObject	public		TComponent
Visible	published	True	TControl
VisibleButtons	published	[btPlay, btPause, btStop, btNext, btPrev, btStep, btBack, btRecord, btEject]	TMediaPlayer
Wait	public		TMediaPlayer
Width	published		TControl
WindowProc	public		TControl

AutoEnable

AutoEnable determines whether the media player automatically enables and disables individual buttons in the component.

property AutoEnable: Boolean;

Description Use *AutoEnable* to automatically enable or disable the control buttons on the media player. If *AutoEnable* is *True*, the media player automatically enables or disables its control buttons. The media player determines which buttons to enable or disable by the current mode specified in the *Mode* property, and the current multimedia device type specified in the *DeviceType* property.

AutoEnable overrides the *EnabledButtons* property. The buttons enabled or disabled automatically by the media player supersede any buttons enabled or disabled with *EnabledButtons*.

If *AutoEnable* is *False*, the media player does not enable or disable buttons. The buttons must enabled or disabled with the *EnabledButtons* property.

The following table shows whether buttons are automatically enabled or disabled for each device mode:

Button	Play	Record	Pause	Stop	Not Open
Back	Enabled	Enabled	Enabled	Enabled	Disabled
Eject	Enabled	Enabled	Enabled	Enabled	Disabled
Next	Enabled	Enabled	Enabled	Enabled	Disabled
Pause	Enabled	Enabled	Enabled	Disabled	Disabled
Play	Disabled	Disabled	Enabled	Enabled	Disabled
Prev	Enabled	Enabled	Enabled	Enabled	Disabled
Record	Disabled	Disabled	Enabled	Enabled	Disabled
Step	Enabled	Enabled	Enabled	Enabled	Disabled
Stop	Enabled	Enabled	Disabled	Disabled	Disabled

See also *AutoButtonSet* method, *DeviceType* property, *EnabledButtons* property, *MMNotify* method, *Mode* property

AutoOpen

AutoOpen determines if the media player is opened automatically when the application is run.

property AutoOpen: Boolean;

Description Use *AutoOpen* to have the media player automatically open at runtime.

If *AutoOpen* is *True*, the media player attempts to open the multimedia device specified by the *DeviceType* property (or *FileName* if *DeviceType* is *dtAutoSelect*) when the form containing the media player component is created at runtime.

If *AutoOpen* is *False*, the device must be opened with a call to the *Open* method.

If an error occurs when opening the device, an exception of type *EMCIDeviceError* exception is raised which contains the error message. Upon

completion, a numerical error code is stored in the *Error* property, and the corresponding error message is stored in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before opening the multimedia device. The *Notify* property determines whether opening the device generates an *OnNotify* event.

See also *DeviceType* property, *EMCIDeviceError* object, *Error* property, *ErrorMessage* property, *FileName* property, *Notify* property, *OnNotify* event, *Wait* property

AutoRewind

AutoRewind sets the current position to the beginning of the previous track.

```
property AutoRewind: Boolean;
```

Description *AutoRewind* determines if the media player control rewinds before playing or recording.

If *AutoRewind* is *True* and the current position is at the end of the medium, *Play* or *StartRecording* moves the current position to the beginning of the medium before playing or recording. If *AutoRewind* is *False*, the user must click the *Prev* button or write code to call *Previous* to move to the beginning.

Note If values have been assigned to *StartPos* or *EndPos* or if the multimedia device uses tracks, *AutoRewind* has no effect on playing or recording. When calling *Play* or *StartRecording*, the current position remains at the end of the medium.

See also *Previous* method

Capabilities

Capabilities determines what the open multimedia device can do.

```
type
  TMPDevCaps = (mpCanStep, mpCanEject, mpCanPlay, mpCanRecord, mpUsesWindows);
  TMPDevCapsSet = set of TMPDevCaps;

property Capabilities: TMPDevCapsSet;
```

Description The various capabilities specified in *Capabilities* are determined when the device is opened with the *Open* method. The following table lists the capabilities a device can have:

Value	Capability
mpCanEject	Can eject media
mpCanPlay	Can play media
mpCanRecord	Can record media
mpCanStep	Can step forward or backward within media
mpUsesWindows	Can use a window for displaying output

Note Currently, there is no way to check whether a device can step forward or backward. *Capabilities* includes *mpCanStep* only if the device type (specified in the *DeviceType* property) is *Animation*, AVI *Video*, *Digital Video*, *Overlay*, or *VCR*.

See also *DeviceType* property, *Open* method

ColoredButtons

ColoredButtons determines which of the buttons on the media player control has color.

```
type
  TMPBtnType = (btPlay, btPause, btStop, btNext, btPrev, btStep, btBack, btRecord, btEject);
  TButtonSet = set of TMPBtnType;

property ColoredButtons: TButtonSet;
```

Description *ColoredButtons* controls which buttons on the media player are colored.

If a button is not colored with *ColoredButtons*, it appears in black-and-white when visible. All media player control buttons are colored by default.

Button	Value	Action
Play	btPlay	Plays the media player
Record	btRecord	Starts recording
Stop	btStop	Stops playing or recording
Next	btNext	Skips to the next track, or to the end if the medium doesn't use tracks
Prev	btPrev	Skips to the previous track, or to the beginning if the medium doesn't use tracks
Step	btStep	Moves forward a number of frames
Back	btBack	Moves backward a number of frames
Pause	btPause	Pauses playing or recording. If already paused when clicked, resumes playing or recording.
Eject	btEject	Ejects the medium

See also *EnabledButtons* property, *VisibleButtons* property

DeviceID

DeviceID specifies the device ID for the currently open multimedia device.

```
property DeviceID: Word;
```

Description The value of *DeviceID* is determined when an device is opened with the *Open* method. If no device is open, *DeviceID* is 0.

See also *Open* method

DeviceType

DeviceType specifies a multimedia device type to open the media player.

```
type TMPDeviceTypes = (dtAutoSelect, dtAVIVideo, dtCDAudio, dtDAT, dtDigitalVideo,
  dtMMMovie, dtOther, dtOverlay, dtScanner, dtSequencer, dtVCR, dtVideodisc, dtWaveAudio);

property DeviceType: TMPDeviceTypes;
```

Description *DeviceType* specifies a multimedia device type to open with the *Open* method.

TMPDeviceTypes type contains the multimedia device types that can be opened by a *TMediaPlayer* component. The valid values for *TMPDeviceTypes* are *dtAutoSelect, dtAVIVideo, dtCDAudio, dtDAT, dtDigitalVideo, dtMMMovie, dtOther, dtOverlay, dtScanner, dtSequencer, dtVCR, dtVideodisc,* or *dtWaveAudio*. The default is *dtAutoSelect*.

If *DeviceType* is *dtAutoSelect*, the device type is determined by the file extension specified in the *FileName* property. If no device type is associated with the extension, the correct device type must be explicitly specified by setting *DeviceType* to a value other than *dtAutoSelect*.

A multimedia device is typically associated with an appropriate file name extension when the device is installed. Associations are specified in the registry or SYSTEM.INI file. See the documentation for the specific device for instructions about how to associate file name extensions with the device.

See also *FileName* property, *Open* method

Display

Display specifies the display window for a multimedia device that uses a window for output.

```
property Display: TWinControl;
```

Description To use *Display*, assign the name of a windowed control such as a form or panel to *Display* to display output in that control.

The default value of *Display* is **nil**, meaning that the device creates its own window to display output. Also, by freeing the control assigned to *Display* after the device has been opened, video output will be in its own default window.

Examples of multimedia devices that use a window to display output are Animation, AVI Video, Digital Video, Overlay, and VCR.

See also *TWinControl* object

DisplayRec

DisplayRect specifies the rectangle area within the form specified in the Display property used to display output from a multimedia device.

```
property DisplayRect: TRect;
```

Description *DisplayRect* is ignored if *Display* is **nil**.

To display output in a specific rectangle area on a form assign a *TRect* record to *DisplayRect*. The *Rect* function can be used to create a *TRect* record.

Examples of multimedia devices that use a window to display output are Animation, AVI Video, Digital Video, Overlay, and VCR.

Media that use a rectangle to display output usually perform best if the default *DisplayRect* size is used. To set *DisplayRect* to the default size, use 0, 0 for the lower right corner. Position the rectangle with the upper left corner.

Note *DisplayRect* can only be set after the media device is opened.

See also *Display* property, *Rect* function, *TRect* record

EnabledButtons

EnabledButtons determines which buttons on the media player are usable.

```
type
  TMPBtnType = (btPlay, btPause, btStop, btNext, btPrev, btStep, btBack, btRecord, btEject);
  TButtonSet = set of TMPBtnType;

property EnabledButtons: TButtonSet;
```

M

Description *EnabledButtons* controls which buttons on the media player are enabled and usable.

An enabled button is colored and usable. A disabled button is dimmed and not usable. If a button is not enabled with *EnabledButtons*, it is disabled. By default, all buttons are enabled.

If the *AutoEnable* property is *True*, *AutoEnable* supersedes *EnabledButtons*. The buttons automatically enabled or disabled by the media player override any buttons enabled or disabled with the *EnabledButtons* property.

Button	Value	Action
Play	btPlay	Plays the media player
Record	btRecord	Starts recording
Stop	btStop	Stops playing or recording
Next	btNext	Skips to the next track, or to the end if the medium doesn't use tracks
Prev	btPrev	Skips to the previous track, or to the beginning if the medium doesn't use tracks
Step	btStep	Moves forward a number of frames
Back	btBack	Moves backward a number of frames
Pause	btPause	Pauses playing or recording. If already paused when clicked, resumes playing or recording.
Eject	btEject	Ejects the medium

See also *ColoredButtons* property, *VisibleButtons* property

EndPos

EndPos specifies the position within the currently loaded medium at which to stop playing or recording.

```
property EndPos: Longint;
```

Description *EndPos* is specified using the current time format, which is specified in the *TimeFormat* property.

The *EndPos* property affects only the next *Play* or *StartRecording* method called after setting *EndPos*. Reset *EndPos* to affect any subsequent calls to *Play* or *StartRecording*.

See also *Play* method, *StartRecording* method, *TimeFormat* property

Error

Error specifies the MCI error code returned by the media control method.

```
property Error: Longint;
```

Description *Error* specifies the MCI error code returned by the most recent media control method (*Back, Close, Eject, Next, Open, Pause, PauseOnly, Play, Previous, StartRecording, Resume, Rewind, Step,* or *Stop*). The *ErrorMessage* property describes the *Error*.

The error code returned by media control methods are the same error code returned by the *mciSendCommand* function. The message describing the error code is stored in the *ErrorMessage* property.

The value of *Error* is zero if the most recent media control method didn't cause an error. If a method results in an error, a value other than zero is stored in *Error*. If the error occurs during the opening of the device, an *EMCIDeviceError* exception occurs.

See also *EMCIDeviceError* exception, *ErrorMessage* property, *Play* method, *StartRecording* method, *TimeFormat* property

ErrorMessage

ErrorMessage describes the error code stored in the *Error* property.

```
property ErrorMessage: string;
```

Description *ErrorMessage* specifies the error message that describes the error code returned from the most recent media control method (*Back, Close, Eject, Next, Open, Pause, PauseOnly, Play, Previous, StartRecording, Resume, Rewind, Step,* or *Stop*).

The error code described by the message is stored in the *Error* property.

See also *Error* property

FileName

FileName specifies the opened or saved media file.

```
property FileName: string;
```

Description *FileName* specifies the media file to be opened by the *Open* method, or the file to save by the *Save* method.

At design time, use a file open dialog box to specify the *FileName* property by clicking the ellipses button (...) in the Object Inspector.

See also *FileName* property, *Open* method, *Save* method

Frames

Frames specifies the number of frames moved forward or backward.

```
property Frames: Longint;
```

Description *Frames* specifies the number of frames the *Step* method steps forward or the *Back* method steps backward.

Frames defaults to ten percent of the length of the currently loaded medium, which is specified by the *Length* property.

Note The definition of frame varies by multimedia device. For display media, a frame is one still image.

See also *Back* method, *Length* property, *Step* method

Length

Length specifies the length of the medium in the open multimedia device.

```
property Length: Longint;
```

Description *Length* is specified using the current time format, which is specified by the *TimeFormat* property.

See also *TimeFormat* property

Mode

Mode indicates the state of the currently open multimedia device.

```
type TMPModes = (mpNotReady, mpStopped, mpPlaying, mpRecording, mpSeeking,
  mpPaused, mpOpen);

property Mode: TMPModes;
```

Description *Mode* specifies the current state or mode of the currently open multimedia device.

The *TMPModes* type defines the modes for a multimedia device used with a *TMediaPlayer*.

The following table lists the possible values for of the *TMPModes* type:

Value	Mode
mpNotReady	Not ready
mpStopped	Stopped
mpPlaying	Playing
mpRecording	Recording
mpSeeking	Seeking
mpPaused	Paused
mpOpen	Open

Notify

Notify determines whether an *OnNotify* event is generated.

```
property Notify: Boolean;
```

Description *Notify* determines whether the next call to a media control method (*Back, Close, Eject, Next, Open, Pause, PauseOnly, Play, Previous, StartRecording, Resume, Rewind, Step,* or *Stop*) generates an *OnNotify* event when the method has completed.

If *Notify* is *True*, the next media control method generates *OnNotify* event upon completion and stores the notification message in the *NotifyValue* property. If *Notify* is *False*, the method does not generate an *OnNotify* event and *NotifyValue* remains unchanged.

Notify affects only the next call to a media control method. After an *OnNotify* event, *Notify* must be reset to affect any subsequent media control methods.

By default, *Play* and *StartRecording* function as if *Notify* is *True*. Set *Notify* to *False* before calling *Play* or *StartRecording* to prevent an *OnNotify* event from being generated when playing or recording has finished. By default, all other media control methods function as if *Notify* is *False*.

Set *Notify* to *True* if the next media control is expected to take a long time, so the application is notified when the media control method has completed. If *Notify* is set to *True*, it is recommended to set *Wait* to *False* so that control returns to the application before the media control method is finished.

Note When trying to resume a device that doesn't support *Resume*, the device is resumed as if the *Play* method was called. If *Notify* was assigned *True* before calling *Resume* (or any other media control method), *Notify* doesn't affect the call to *Resume*. *Resume* does not generate an *OnNotify* event upon completion, and *NotifyValue* remains unchanged.

See also *NotifyValue* property, *OnNotify* event, *Resume* method, *Wait* property

NotifyValue

NotifyValue indicates the result of the last media control method that requested a notification.

```
type TMPNotifyValues = (nvSuccessful, nvSuperseded, nvAborted, nvFailure);
property NotifyValue: TMPNotifyValues;
```

Description *NotifyValue* reports the result of the last media control method (*Back, Close, Eject, Next, Open, Pause, PauseOnly, Play, Previous, StartRecording, Resume, Rewind, Step,* or *Stop*) that requested a notification.

To request notification, set *Notify* to *True* before calling a media control method.

The *TMPNotifyValues* type defines the notification values for a multimedia device used with a *TMediaPlayer*. The possible values for the *TMPNotifyValues* type are as follows.

Value	Result
nvSuccessful	Command completed successfully
nvSuperseded	Command was superseded by another command
nvAborted	Command was aborted by the user
nvFailure	Command failed

M

Position

Position specifies the current position within the currently loaded medium.

```
property Position: Longint;
```

Description The value of *Position* is specified according to the current time format, which is specified in the *TimeFormat* property.

Position defaults to the beginning of the medium. If the medium supports multiple tracks, *Position* defaults to the beginning of the first track.

See also *TimeFormat* property

Shareable

Shareable determines whether more than one application can share a multimedia device.

```
property Shareable: Boolean;
```

Description If *Shareable* is *False*, no other components or applications can access the device. If *Shareable* is *True*, more than one component or application can access the device. *Shareable* defaults to *False*.

Be sure to set *Shareable* before opening a device. Some devices aren't shareable. If you set *Shareable* to *True* and try to open a device that isn't shareable by more than one application, the *Open* method fails and the error code is returned to the *Error* property.

See also *Error* property, *Open* method

Start

Start specifies the starting position within the currently loaded medium.

```
property Start: Longint;
```

Description *Start* is the beginning of the medium for devices that don't use tracks, or the beginning of the first track for devices that use tracks. *Start* is defined when a multimedia device is opened with the *Open* method. *Start* is specified according to the current time format, which is stored in the *TimeFormat* property. *Start* is read-only at runtime and is unavailable at design time.

See also *Open* method, *TimeFormat* property

StartPos

StartPos specifies the position within the currently loaded medium from which to begin playing or recording.

```
property StartPos: Longint;
```

Description *StartPos* is specified using the current time format, which is specified in the *TimeFormat* property.

The *StartPos* property affects only the next *Play* or *StartRecording* method called after setting *StartPos*. Reset must be set to *StartPos* to affect any subsequent calls to *Play* or *StartRecording*.

StartPos does not affect the current position of the medium (specified in the Position property) until the next *Play* or *StartRecording* method is called.

See also *Play* method, *StartRecording* method, *TimeFormat* property

TimeFormat

TimeFormat determines the format used to specify position information.

```
type TMPTimeFormats = (tfMilliseconds, tfHMS, tfMSF, tfFrames, tfSMPTE24, tfSMPTE25,
  tfSMPTE30, tfSMPTE30Drop, tfBytes, tfSamples, tfTMSF);

property TimeFormat: TMPTimeFormats;
```

Description *TimeFormat* determines how the *StartPos, Length, Position, Start,* and *EndPos* properties are interpreted. For example, if *Position* is 180 and *TimeFormat* is *tfMilliseconds,* the current position is 180 milliseconds into the medium. If *Position* is 180 and *TimeFormat* is *tfMSF,* the current position is 180 minutes into the medium.

Not all formats are supported by every device. If you try to set an unsupported format, the assignment is ignored.

The current timing information is always passed in a 4-byte integer. In some formats, the timing information returned is not really one integer, but single bytes of information packed in the long integer.

The *TMPTimeFormats* type defines the time formats for a multimedia device used with a *TMediaPlayer*. The following table lists the possible values for the TMP*TimeFormats* type:

Value	Time format
tfMilliseconds	Milliseconds are stored as a 4-byte integer variable.
tfHMS	Hours, minutes, and seconds packed into a 4-byte integer. From least significant to most significant byte, the data values are Hours (least significant byte) Minutes Seconds Unused (most significant byte)
tfMSF	Minutes, seconds, and frames packed into a 4-byte integer. From least significant to most significant byte, the data values are Minutes (least significant byte) Seconds Frames Unused (most significant byte)
tfFrames	Frames are stored as a 4-byte integer variable.
tfSMPTE24	24-frame SMPTE packs values in a 4-byte variable. From least significant to most significant byte, the data values are Hours (least significant byte) Minutes Seconds Frames (most significant byte) SMPTE (Society of Motion Picture and Television Engineers) time is an absolute time format expressed in hours, minutes, seconds, and frames. The standard SMPTE division types are 24, 25, and 30 frames per second.
tfSMPTE25	25-frame SMPTE packs data into a 4-byte variable in the same order as 24-frame SMPTE.
tfSMPTE30	30-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE.
tfSMPTE30Drop	30-drop-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE.
tfBytes	Bytes are stored as a 4-byte integer variable.
tfSamples	Samples are stored as a 4-byte integer variable.
tfTMSF	Tracks, minutes, seconds, and frames are packed in the 4-byte variable. The data values are Tracks (least significant byte) Minutes Seconds Frames (most significant byte)

Note that MCI uses continuous track numbering.

Note Functions provided with MCI to help decoding the 4-byte integer specified in a given time format are documented under MCI Macros for Encoding and Decoding Time Data in the MMSYSTEM.HLP Help file.

See also *EndPos* property, *Length* property, *Position* property, *Start* property, *StartPos* property

TrackLength

TrackLength reports the length of the track.

```
property TrackLength[TrackNum: Integer]: Longint;
```

Description *TrackLength* reports the length of the track specified by the *TrackNum* index.

The value of *TrackLength* is specified according to the current time format, which is specified in the *TimeFormat* property.

See also *TimeFormat* property

TrackPosition

TrackPosition reports the starting position of the track.

```
property TrackPosition[TrackNum: Integer]: Longint;
```

Description *TrackPosition* reports the starting position of the track specified by the *TrackNum* index.

The value of *TrackPosition* is specified according to the current time format, which is specified in the *TimeFormat* property.

See also *TimeFormat* property

Tracks

Tracks specifies the number of playable tracks on the open multimedia device.

```
property Tracks: Longint;
```

Description *Tracks* indicates how many of playable tracks are available on the open multimedia device.

Tracks is undefined for devices that don't use tracks.

VisibleButtons

VisibleButtons indicates which of the buttons on the media player are visible.

```
type
  TMPBtnType = (btPlay, btPause, btStop, btNext, btPrev, btStep, btBack, btRecord, btEject);
  TButtonSet = set of TMPBtnType;

property VisibleButtons: TButtonSet;
```

Description *VisibleButtons* determines the buttons on the media player that are visible. If a button is not made visible with *VisibleButtons*, it does not appear on the media player control. By default, all buttons are visible when a media player component is added to a form.

Button	Value	Action
Play	btPlay	Plays the media player
Record	btRecord	Starts recording
Stop	btStop	Stops playing or recording
Next	btNext	Skips to the next track, or to the end if the medium doesn't use tracks
Prev	btPrev	Skips to the previous track, or to the beginning if the medium doesn't use tracks
Step	btStep	Moves forward a number of frames
Back	btBack	Moves backward a number of frames
Pause	btPause	Pauses playing or recording. If already paused when clicked, resumes playing or recording.
Eject	btEject	Ejects the medium

See also *ColoredButtons* property, *EnabledButtons* property

Wait

Wait determines whether a media control method returns control to the application only after it has been completed.

```
property Wait: Boolean;
```

Description *Wait* determines whether a media control method (*Back, Close, Eject, Next, Open, Pause, PauseOnly, Play, Previous, StartRecording, Resume, Rewind, Step,* or *Stop*) returns control to the application only after it has been completed.

Wait is unavailable at design time.

If *Wait* is *True*, the media player component waits until the next media control method has completed before returning control to the application. If *Wait* is *False*, the application won't wait for the next media control method to finish before continuing.

Wait affects only the next media control method called after setting *Wait*. *Wait* must be reset to affect any subsequent call to a media control method.

By default, *Play* and *StartRecording* function as if *Wait* is *False*. *Wait* must be set to *True* before calling *Play* or *StartRecording* to prevent control from returning to the application before playing or recording has finished. By default, all other media control methods function as if *Wait* is *True*.

Note *Wait* is usually set to *False* only if the next media control is expected to take a long time, so that the application can execute other code before the media control method has completed. If *Wait* is set to *False*, it is recommended to set *Notify* to *True* so the application is notified when the media control method completes.

Events

The following table lists the events of *TMediaPlayer*:

Event	Visibility	Documented in
OnClick	published	TMediaPlayer
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnNotify	published	TMediaPlayer
OnPostClick	published	TMediaPlayer

OnClick

OnClick occurs when the user presses and releases the mouse button over one of the control buttons of the media player control, or when the user presses *Spacebar* while the media player control has focus.

```
type EMPNotify = procedure (Sender: TObject; Button: TMPBtnType; var DoDefault: Boolean)
  of object;

property OnClick: EMPNotify;
```

Description When the media player control has focus and the Spacebar is pressed, the user can use the Left Arrow or Right Arrow keys to select which control button to click.

The *EMPNotify* type is a method pointer that is called when an *OnClick* event for a *TMediaPlayer* components occurs. The *Button* argument can be one of the following values: *btBack*, *btEject*, *btNext*, *btPause*, *btPlay*, *btPrev*, *btRecord*, *btStep*, or *btStop*.

The default value of the *DoDefault* argument is *True*. If *DoDefault* is *True*, the media player control calls the method that corresponds to the clicked button. For example, if the user clicks the *Play* button (*btPlay*), the *Play* method is called.

If *DoDefault* is *False*, the user must write code that executes when a media player control button is clicked in the *OnClick* event handler. The following

table lists the default methods corresponding to the media player control buttons:

Control button	Button value	Method called
Play	btPlay	Play
Record	btRecord	StartRecording
Stop	btStop	Stop
Next	btNext	Next
Prev	btPrev	Previous
Step	btStep	Step
Back	btBack	Back
Pause	btPause	Pause
Eject	btEject	Eject

See also *Click* method

OnNotify

OnNotify occurs upon the completion of a media control method.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnNotify: TNotifyEvent;
```

Description *OnNotify* occurs upon the completion of a media control method (*Back, Close, Eject, Next, Open, Pause, PauseOnly, Play, Previous, Resume, Rewind, StartRecording, Step,* or *Stop*) when the *Notify* property is set to *True* before the call to the media control method. After an *OnNotify* event, the *Notify* property must be reset to *True* for the next *OnNotify* event to occur.

See also *DoNotify* method, *MMNotify* method, *Notify* property

OnPostClick

OnPostClick occurs after the *OnClick* event handler is called.

```
type EMPPostNotify = procedure (Sender: TObject; Button: TMPBtnType) of object;

property OnPostClick: EMPPostNotify;
```

Description *OnPostClick* is generated after the code of the *OnClick* event handler has been called. If *Wait* is *True* when the media player was clicked, *OnPostClick* won't be called until the completion of the *OnClick* code. If *Wait* is *False*, control can return to the application before completion of the *OnClick* code; therefore, the *OnPostClick* event may occur before the actions initiated by the *OnClick* event have completed.

For example, if the user clicks the *Play* button and the *DoDefault* parameter of the *OnClick* event handler for the media player is *True*, the media is played. If the media is long enough, it will still be playing when the *OnPostClick* event

is generated if *Wait* is *True*. If *Wait* is *False*, however, *OnPostClick* won't occur until the media has finished playing.

The *EMPPostNotify* type is a method that is called when an *OnPostClick* event for a *TMediaPlayer* component occurs. The *Button* argument can be one of the following values: *btBack*, *btEject*, *btNext*, *btPause*, *btPlay*, *btPrev*, *btRecord*, *btStep*, or *btStop*.

See also *OnClick* event, *PostClick* method, *Wait* property

Methods

The following table lists the methods of *TMediaPlayer*:

Method	Visibility	Documented in
Assign	public	TPersistent
AutoButtonSet	protected	TMediaPlayer
Back	public	TMediaPlayer
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Click	protected	TMediaPlayer
ClientToScreen	public	TControl
Close	public	TMediaPlayer
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TMediaPlayer
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TMediaPlayer
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DoNotify	protected	TMediaPlayer
DragDrop	public	TControl
Dragging	public	TControl
Eject	public	TMediaPlayer

Method	Visibility	Documented in
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
Loaded	protected	TMediaPlayer
MethodAddress	public	TObject
MethodName	public	TObject
MMNotify	protected	TMediaPlayer
NewInstance	public	TObject
Next	public	TMediaPlayer
Notification	protected	TMediaPlayer
Open	public	TMediaPlayer
Paint	protected	TMediaPlayer
PaintTo	public	TWinControl
Pause	public	TMediaPlayer
PauseOnly	public	TMediaPlayer
Perform	public	TControl
Play	public	TMediaPlayer
PostClick	protected	TMediaPlayer

Method	Visibility	Documented in
Previous	public	TMediaPlayer
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
Resume	public	TMediaPlayer
Rewind	public	TMediaPlayer
SafeCallException	public	TComponent
Save	public	TMediaPlayer
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
StartRecording	public	TMediaPlayer
Step	public	TMediaPlayer
Stop	public	TMediaPlayer
Update	public	TWinControl
UpdateControlState	public	TWinControl
Updated	protected	TMediaPlayer

AutoButtonSet

AutoButtonSet controls the enabling of the media player's buttons.

```
type
  TMPBtnType = (btPlay, btPause, btStop, btNext, btPrev, btStep, btBack, btRecord, btEject);
  TButtonSet = set of TMPBtnType;

procedure AutoButtonSet(Btn: TMPBtnType); dynamic;
```

Description *AutoButtonSet* implements the automatic enabling and disabling of the media player's buttons when the *AutoEnable* property is *True*. The *Btn* parameter indicates which of the media player's buttons the user pressed.

TMPBtnType is the type of the *Btn* parameter. The values for *TMPBtnType* are as follows:

Button	Value	Action
Play	btPlay	Plays the media player
Record	btRecord	Starts recording

Button	Value	Action
Stop	btStop	Stops playing or recording
Next	btNext	Skips to the next track, or to the end if the medium doesn't use tracks
Prev	btPrev	Skips to the previous track, or to the beginning if the medium doesn't use tracks
Step	btStep	Moves forward a number of frames
Back	btBack	Moves backward a number of frames
Pause	btPause	Pauses playing or recording. If already paused when clicked, resumes playing or recording.
Eject	btEject	Ejects the medium

See also *AutoEnable* property

Back

Back steps backward a number of frames (determined by the value of the Frames property) in the currently loaded medium.

```
procedure Back;
```

Description *Back* is called when the *Back* button on the media player control is clicked at runtime.

Upon completion, *Back* stores a numerical error code in the *Error* property and the corresponding error message in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before the Back method has been completed. The *Notify* property determines whether *Back* generates an *OnNotify* event.

See also *Error* property, *ErrorMessage* property, *Notify* property, *OnNotify* event, *Wait* property

Click

Click determines the action that takes place when an *OnClick* event occurs.

```
procedure Click(Button: TMPBtnType; var DoDefault: Boolean); dynamic;
```

Description *Click* for media-player components is the implementation method for the *OnClick* event. By default, *Click* does nothing other than call any event handler attached to the media player's *OnClick* event.*Click* can be overridden in descendant types to customize responses to clicks.

See also *OnClick* event

Close

Close closes the open multimedia device.

```
procedure Close;
```

Description Upon completion, *Close* stores a numerical error code in the *Error* property, and the corresponding error message in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before the *Close* method is completed. The *Notify* property determines whether *Close* generates an *OnNotify* event.

Close is called automatically when the application is terminated.

See also *Error* property, *ErrorMessage* property, *Notify* property, *OnNotify* event, *Wait* property

Create

Create instantiates a media player object.

```
constructor Create(AOwner: TComponent); override;
```

Description Call *Create* to instantiate a media player at runtime. For media players created at design time, *Create* is called automatically.

Create allocates memory for a media player, and calls the inherited *Create*.

Then it loads the bitmaps used for its buttons and sets initial values for the media player including setting the *AutoEnable*, *AutoRewind*, *Colored*, *Enabled*, and *Visible* properties to *True*, *AutoOpen* to *False*, and *DeviceType* to *dtAutoSelect*.

See also *TCustomControl.Create* method

Destroy

Destroy disposes of the instance of the media player.

```
destructor Destroy; override;
```

Description Do not call *Destroy* directly in an application. Instead, an application should call *Free*. *Free* verifies that the media player object has not already been freed before it calls *Destroy*.

Destroy ensures that no device is open before it destroys the bitmaps, used for the media player's buttons, and then calls the inherited *Destroy*.

See also *TCustomControl.Destroy* method

DoNotify

DoNotify determines the action that takes place when an *OnNotify* event occurs.

procedure DoNotify; **dynamic**;

Description *DoNotify* is the implementation method for a media-player component's *OnNotify* event. The media player's *MMNotify* method calls *DoNotify* after correcting the enabling and disabling of buttons and setting internal flags from the values passed in the notification message parameters.

By default, *DoNotify* does nothing except call any event handler attached to the media player's OnNotify event. *DoNotify* can be overridden to provide other responses in addition to the inherited event-handler call.

See also *MMNotify* method, *OnNotify* event

Eject

Eject forcefully releases the loaded medium from the open multimedia device.

procedure Eject;

Description *Eject* ejects the loaded medium from the open multimedia device.

Eject is called when the *Eject* button on the media player control is clicked at runtime.

Upon completion, *Eject* stores a numerical error code in the *Error* property, and the corresponding error message in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before the *Eject* method has been completed. The *Notify* property determines whether *Eject* generates an *OnNotify* event.

See also *Error* property, *ErrorMessage* property, *Notify* property, *OnNotify* event, *Wait* property

Loaded

Loaded provides an opportunity for initialization after the media player has been loaded from a stream.

procedure Loaded; **override**;

Description *Loaded* provides an opportunity for a component to initialize itself after all its parts have loaded from a stream. When a Delphi application loads a form from its form file, for example, it first constructs the form component by calling its *Create* constructor, then reading its property values from the form file, which is a stream. After reading all the property values for all the components, Delphi calls the *Loaded* methods of each component in the order the components were created. This gives the components a chance to

initialize any data that depends on the values of other components or other parts of itself.

See also *Create* method, *TFiler* object, *TStream* object

Next

Next moves to the beginning of the next track of the currently loaded medium.

```
procedure Next;
```

Description If the current position is at the last track when *Next* is called, *Next* makes the current position the beginning of the last track. If the multimedia device doesn't use tracks, *Next* goes to the end of the medium. *Next* is called when the *Next* button on the media player control is clicked at runtime.

Upon completion, *Next* stores a numerical error code in the *Error* property, and the corresponding error message in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before the *Next* method has completed. The *Notify* property determines whether *Next* generates an *OnNotify* event.

See also *Error* property, *ErrorMessage* property, *Notify* property, *OnNotify* event, *Wait* property

MMNotify

MMNotify responds to a notification that the MCI device has completed an operation.

```
procedure MMNotify(var Message: TMessage); message MM_MCINOTIFY;
```

Description *MMNotify* responds to a notification message from Windows that a multimedia command completed by updating button states if the *AutoEnable* property is *True*, setting internal flags to reflect the outcome of the command, then calling the *DoNotify* method to trigger the *OnNotify* event and any other special responses.

See also *AutoEnable* property, *DoNotify* method, *OnNotify* event

Notification

Notification notifies the component that the component specified by *AComponent* is about to be inserted or removed, as specified by *Operation*.

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

Description By default, components pass along the notification to their owned components, if any. *Notification* calls the inherited *Notification* and then, if the Display window is deleted at runtime, it sets the video display back to original window.

See also *TControl.Notification* method

Open

Open opens a multimedia device.

```
procedure Open;
```

Description Use *Open* to open a multimedia device. The multimedia device type must be specified in the *DeviceType* property before a device can be opened.

Upon completion, *Open* stores a numerical error code in the *Error* property, and the corresponding error message in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before the *Open* method is completed. The *Notify* property determines whether *Open* generates an *OnNotify* event.

See also *Error* property, *ErrorMessage* property, *Notify* property, *OnNotify* event, *Wait* property

Paint

Paint responds to a WM_PAINT Windows message.

```
procedure Paint; override;
```

Description *Paint* renders the image of the media player. Custom controls, such as the medial player component, respond to WM_PAINT messages by initializing the control's canvas and calling *Paint*.

When creating a descendant media player class, *Paint* can be overridden to draw the image of the corresponding media player object.

PostClick

PostClick determines the action that takes place when an *OnPostClick* event occurs.

```
procedure PostClick(Button: TMPBtnType); dynamic;
```

Description *PostClick* is the implementation method for the *OnPostClick* event. By default, *PostClick* does nothing other than call any event handler attached to the media player's *OnPostClick* event. *PostClick* can be overridden in descendant types to customize responses to clicks.

See also *OnPostClick* event

Pause

Pause pauses the open multimedia device.

```
procedure Pause;
```

Description If the device is already paused when *Pause* is called, the device resumes playing or recording by calling the *Resume* method. *Pause* is called when the *Pause* button on the media player control is clicked at runtime.

Upon completion, *Pause* stores a numerical error code in the *Error* property and the corresponding error message in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before the *Pause* method has completed. The *Notify* property determines whether *Pause* generates an *OnNotify* event.

See also *Error* property, *ErrorMessage* property, *Notify* property, *OnNotify* event, *Resume* method, *Wait* property

PauseOnly

PauseOnly pauses the open multimedia device.

```
procedure PauseOnly;
```

Description If the device is already paused when *PauseOnly* is called, the device will remain paused.

Upon completion, *PauseOnly* stores a numerical error code in the *Error* property and the corresponding error message in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before the *PauseOnly* method has completed. The *Notify* property determines whether *PauseOnly* generates an *OnNotify* event.

See also *Error* property, *ErrorMessage* property, *Notify* property, *OnNotify* event, *Wait* property

Play

Play plays the media loaded in the open multimedia device.

```
procedure Play;
```

Description *Play* is called when the *Play* button on the media player control is clicked at runtime.

Upon completion, *Play* stores a numerical error code in the *Error* property and the corresponding error message in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before the *Play* method has completed. The *Notify* property determines whether *Play* generates an *OnNotify* event.

If the *StartPos* property is set, playing starts at the position specified in *StartPos*. Otherwise, playing starts at the current position, specified in the *Position* property. Similarly, if the *EndPos* property is set, playing stops at the position specified in *EndPos*. Otherwise, playing stops at the end of the medium.

Whether the medium (specified in the *Position* property) is rewound before playing starts depends on the *AutoRewind* property.

See also *AutoRewind* property, *EndPos* property, *Error* property, *ErrorMessage* property, *Notify* property, *OnNotify* event, *Position* property, *Resume* method, *StartPos* property, *Wait* property

Previous

Previous sets the current position to the beginning of the previous track if the position was at the beginning of a track when *Previous* was called.

```
procedure Previous;
```

Description If the position is at the first track or somewhere other than the beginning of a track when *Previous* was called, *Previous* sets the current position to the beginning of the current track. If the device doesn't use tracks, *Previous* sets the current position to the beginning of the medium, which is specified in the *Start* property. *Previous* is called when the *Previous* button on the media player control is clicked at runtime.

Upon completion, *Previous* stores a numerical error code in the *Error* property and the corresponding error message in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before the *Previous* method has completed. The *Notify* property determines whether *Previous* generates an *OnNotify* event.

See also *Error* property, *ErrorMessage* property, *Notify* property, *OnNotify* event, *Start* property, *Wait* property

Resume

Resume resumes playing or recording the currently paused multimedia device.

```
procedure Resume;
```

Description *Resume* is called when the *Pause* button on the media player control is clicked at runtime, when the device is paused.

Upon completion, *Resume* stores a numerical error code in the *Error* property, and the corresponding error message in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before the *Resume* method has completed. The *Notify* property determines whether *Resume* generates an *OnNotify* event.

See also *Error* property, *ErrorMessage* property, *Notify* property, *OnNotify* event, *Wait* property

Rewind

Rewind sets the current position to the beginning of the medium, which is stored in the *Start* property.

procedure Rewind;

Description Upon completion, *Rewind* stores a numerical error code in the *Error* property, and the corresponding error message in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before the *Rewind* method has completed. The *Notify* property determines whether *Rewind* generates an *OnNotify* event.

See also *Error* property, *ErrorMessage* property, *Notify* property, *OnNotify* event, *Start* property, *Wait* property

Save

Save saves the currently loaded medium to the file specified in the *FileName* property.

procedure Save;

Description *Save* is ignored for devices that don't use media stored in files (videodiscs, for example).

Upon completion, *Save* stores a numerical error code in the *Error* property, and the corresponding error message in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before the *Save* method has completed. The *Notify* property determines whether *Save* generates an *OnNotify* event.

See also *Error* property, *ErrorMessage* property, *FileName* property, *Notify* property, *OnNotify* event, *Wait* property

StartRecording

StartRecording begins recording from the current *Position* or from the position specified in *StartPos*.

procedure StartRecording;

Description *StartRecording* is called when the *Record* button on the media player control is clicked at runtime.

Upon completion, *StartRecording* stores a numerical error code in the *Error* property and the corresponding error message in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before the *StartRecording* method has completed. The *Notify* property determines whether *StartRecording* generates an *OnNotify* event.

By default, the *Notify* property becomes *True*, and the *Wait* property becomes *False* upon completion of the *StartRecording* method. However, if these properties have been set to specific values prior to calling *StartRecording*, they remain unchanged.

See also *Error* property, *ErrorMessage* property, *Notify* property, *OnNotify* event, *Position* property, *StartPos* property, *Wait* property

Step

Step moves forward a number of frames (determined by the *Frames* property) in the currently loaded medium.

```
procedure Step;
```

Description *Step* is called when the *Step* button on the media player control is clicked at runtime.

Upon completion, *Step* stores a numerical error code in the *Error* property and the corresponding error message in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before the *Step* method has completed. The *Notify* property determines whether *Step* generates an *OnNotify* event.

See also *Error* property, *ErrorMessage* property, *Frames* property, *Notify* property, *OnNotify* event, *Wait* property

Stop

Stop halts playing or recording.

```
procedure Stop;
```

Description *Stop* is called when the *Stop* button on the media player control is clicked at runtime.

Upon completion, *Stop* stores a numerical error code in the *Error* property, and the corresponding error message in the *ErrorMessage* property.

The *Wait* property determines whether control is returned to the application before the *Stop* method has completed. The *Notify* property determines whether *Stop* generates an *OnNotify* event.

See also *Error* property, *ErrorMessage* property, *FileName* property, *Notify* property, *OnNotify* event, *Wait* property

Updated

Updated signals that the media player has finished updating.

```
procedure Updated; override;
```

Description *Updated* calls the inherited *Updated* before it modifies the layout of the media player buttons forces a repaint.

See also *TComponent.Updated* method

TMemo

Unit: StdCtrls | Palette: Standard

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomEdit • TCustomMemo • TMemo

TMemo is a wrapper for a windows multiline edit control.

Description Use *TMemo* to put a standard Windows multiline edit control on a form. Multiline edit boxes allow the user to enter more than one line of text. They are appropriate for representing lengthy information.

TMemo implements the generic behavior introduced in *TCustomMemo*. *TMemo* publishes many of the properties inherited from *TCustomMemo*, but does not introduce any new behavior.

See also *TDBMemo* object, *TEdit* object, *TRichEdit* object

Properties

The following table lists the properties of *TMemo*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
Alignment	published	taLeftJustify	TCustomMemo
BorderStyle	published	bsSingle	TCustomEdit
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl

Property	Visibility	Default	Documented in
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
HideSelection	published	True	TCustomEdit
Hint	published		TControl
ImeMode	published	imDontCare	TWinControl
ImeName	published		TWinControl
Left	published		TControl
Lines	published		TCustomMemo
MaxLength	published	0	TCustomEdit
Modified	public		TCustomEdit
Name	published		TControl
OEMConvert	published	False	TCustomEdit
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ReadOnly	published	False	TCustomEdit
ScrollBars	published	ssNone	TCustomMemo
SelLength	public		TCustomEdit
SelStart	public		TCustomEdit
SelText	public		TCustomEdit
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent
Text	public		TControl
Top	published		TControl

Property	Visibility	Default	Documented in
VCLComObject	public		TComponent
Visible	published	True	TControl
WantReturns	published	True	TCustomMemo
WantTabs	published	False	TCustomMemo
Width	published		TControl
WindowProc	public		TControl
WordWrap	published	True	TCustomMemo

Events

The following table lists the events of *TMemo*:

Event	Visibility	Documented in
OnChange	published	TCustomEdit
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TMemo*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject

Method	Visibility	Documented in
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TCustomEdit
ClearSelection	public	TCustomEdit
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
CopyToClipboard	public	TCustomEdit
Create	public	TCustomMemo
CreateParented	public	TWinControl
CutToClipboard	public	TCustomEdit
DefaultHandler	public	TObject
Destroy	public	TCustomMemo
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetSelTextBuf	public	TCustomEdit
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent

Method	Visibility	Documented in
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
PasteFromClipboard	public	TCustomEdit
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SelectAll	public	TCustomEdit
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetSelTextBuf	public	TCustomEdit
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

TMemoField

Unit: DB

TObject • TPersistent • TComponent • TField • TBlobField • TMemoField

The *TMemoField* component represents a memo field in a dataset.

Description Memo fields are a form of binary large object (BLOB) field where the data consists of an arbitrarily large string of text. *TMemoField* differs from its immediate ancestor *TBlobField* only in the value of the *DataType* property. As

a descendant of *TBlobField*, *TMemoField* includes many properties, methods, and events that are useful for managing the value and properties of a BLOB field in a database.

See also *TDataSet object*, *TGraphicField object*, *TDBMemo object*

Properties

The following table lists the properties of *TMemoField*:

Property	Visibility	Default	Documented in
Alignment	published	taLeftJustify	TField
AsBoolean	public		TField
AsCurrency	public		TField
AsDateTime	public		TField
AsFloat	public		TField
AsInteger	public		TField
AsString	public		TBlobField
AsVariant	public		TBlobField
AttributeSet	public		TField
BlobSize	public, read-only		TBlobField
BlobType	published		TBlobField
Calculated	public	False	TField
CanModify	public, read-only		TField
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ConstraintErrorMessage	published		TField
CurValue	public, read-only		TField
CustomConstraint	published		TField
DataSet	public		TField
DataSize	public, read-only		TField
DataType	public, read-only		TField
DefaultExpression	published		TField
DesignInfo	public		TComponent
DisplayLabel	published		TField
DisplayName	public, read-only		TField
DisplayText	public, read-only		TField
DisplayWidth	published		TField
EditMask	public		TField
EditMaskPtr	public, read-only		TField

Property	Visibility	Default	Documented in
FieldKind	published	fkData	TField
FieldName	published		TField
FieldNo	public, read-only		TField
HasConstraints	published, read-only		TField
ImportedConstraint	published		TField
Index	published		TField
IsIndexField	public, read-only		TField
IsNull	public, read-only		TBlobField
KeyFields	published		TField
Lookup	public		TField
LookupCache	published	False	TField
LookupDataSet	published		TField
LookupKeyFields	published		TField
LookupList	public, read-only		TField
LookupResultField	published		TField
Modified	public		TBlobField
Name	published		TComponent
NewValue	public		TField
Offset	public, read-only		TField
OldValue	public, read-only		TField
Origin	published		TField
Owner	public, read-only		TComponent
ReadOnly	published	False	TField
Required	published	False	TField
Size	published	0	TField
Tag	published	0	TComponent
Text	public		TField
Transliterate	published	True	TBlobField
ValidChars	public		TField
Value	public		TBlobField
VCLComObject	public		TComponent
Visible	published	True	TField

Events

The following table lists the events of *TMemoField*:

Event	Visibility	Documented in
OnChange	published	TField
OnGetText	published	TField

Event	Visibility	Documented in
OnSetText	published	TField
OnValidate	published	TField

Methods

The following table lists the methods of *TMemoField*:

Method	Visibility	Documented in
Assign	public	TBlobField
AssignValue	public	TField
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TBlobField
Create	public	TMemoField
DefaultHandler	public	TObject
Destroy	public	TField
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
FocusControl	public	TField
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetData	public	TField
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject

Method	Visibility	Documented in
IsBlob	public	TBlobField
IsValidChar	public	TField
LoadFromFile	public	TBlobField
LoadFromStream	public	TBlobField
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RefreshLookupList	public	TField
RemoveComponent	public	TComponent
SafeCallException	public	TComponent
SaveToFile	public	TBlobField
SaveToStream	public	TBlobField
SetData	public	TField
SetFieldType	public	TBlobField
SetText	public	TBlobField
Validate	public	TField

Create

The Create method creates an instance of *TMemoField*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to instantiate a *TMemoField* object at runtime. For *TMemoField* objects placed on forms or data modules at design time, *Create* is called automatically.

After calling the inherited constructor, *Create* sets the *DataType* to *ftMemo*, and initializes the *Transliterate* property to *True*.

See also *DataType property*, *TField.Create* method, *Transliterate* property

TMenu

Unit: Menus

TObject • TPersistent • TComponent • TMenu

The *TMenu* component is the base type for menu components such as *TMainMenu* and *TPopupMenu*.

Description Use *TMenu* as a base class when defining a component that represents a collection of menu items. *TMenu* introduces properties and methods to create a Windows menu that represents a set of menu items, dispatch commands associated with those menu items, and locate the help context ID for any of those menu items.

See also *TMainMenu* object, *TMenuItem* object, *TPopupMenu* object

Properties

The following table lists the properties of *TMenu*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DesignInfo	public		TComponent
Handle	public, read-only		TMenu
Items	published, read-only		TMenu
Name	published		TComponent
Owner	public, read-only		TComponent
Tag	published	0	TComponent
VCLComObject	public		TComponent
WindowHandle	public		TMenu

M

Handle

Handle is the Windows menu handle for the menu.

property Handle: HMENU;

Description Use *Handle* to call a Windows API function that requires a menu handle. *Handle* is the handle for the menu that contains only the items described by the *Items* property. If no other menus have been merged with this menu, this is the same as the *WindowHandle* property.

See also *WindowHandle* property

Items

Items is a *TMenuItem* object that describes the elements of the menu.

property Items: TMenuItem;

Description Use *Items* to access information about the elements in the menu. *Items* is a single *TMenuItem* object that describes the elements of the menu in its own *Items* property. Because *Items* is the default property of *TMenuItem*, the *Items* property of *TMenu* can be treated as an indexed array of menu items that describe the individual items in the menu. That is, instead of writing

```
FirstItem := Menu1.Items.Items[0];
```

use

```
FirstItem := Menu1.Items[0];
```

Set the *Items* property at design time by clicking on the *Items* property in the Property Inspector. The Menu Designer automatically inserts a menu item. As each menu item is entered, outlined regions appear at the possible locations of additional menu items. Click in those regions to specify additional menu items. At runtime, create a *TMenuItem* that describes the elements of the menu programmatically, and set the *Items* property of the menu.

See also *TMenuItem* object, *TMenuItem.Items* property

WindowHandle

WindowHandle is handle of the window that uses the menu.

property WindowHandle: HWnd;

Description Read *WindowHandle* to get the window handle of the control that receives WM_COMMAND messages when the user clicks on an item in the menu. When menu items are selected, use this handle in an *OnClick* event handler to send messages to the control that uses the menu or to make Windows API calls that affect the control.

Set *WindowHandle* to associate the menu with a control.

See also *DispatchCommand* method, *TMenuItem.Command* property, *TMenuItem.OnClick* event

Methods

The following table lists the methods of *TMenu*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TMenu
DefaultHandler	public	TObject
Destroy	public	TMenu
DestroyComponents	public	TComponent
Destroying	public	TComponent

Method	Visibility	Documented in
Dispatch	public	TObject
DispatchCommand	public	TMenu
DispatchPopup	public	TMenu
FieldAddress	public	TObject
FindComponent	public	TComponent
FindItem	public	TMenu
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetHelpContext	public	TMenu
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
IsShortCut	public	TMenu
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

M

Create

Create creates an instance of *TMenu*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to create a menu at runtime. Menus placed on forms at design time are created automatically. *Create* calls the inherited *Create* method, then sets the initial values for the menu component, including creating an *Items* property.

See also *Items* property

Destroy

Destroy destroys the menu component.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the menu has not already been freed, and only then calls *Destroy*. *Destroy* frees the *Items* property, and then calls the inherited *Destroy* method.

See also *Free* method, *Items* property

DispatchCommand

DispatchCommand calls the *Click* method of the menu item with the specified value as its *Command* property.

function DispatchCommand(ACommand: Word): Boolean;

Description Call *DispatchCommand* to simulate a click on the menu item associated with a command ID. If *DispatchCommand* locates a menu item that matches the command ID, it calls the *Click* method for that menu item and returns *True*. If there is no such item, *DispatchCommand* returns *False*.

See also *DispatchPopup* method, *IsShortCut* method, *Items* property, *TMenuItem.Click* method

DispatchPopup

DispatchPopup dispatches a pop-up menu command.

function DispatchPopup(AHandle: HMENU): Boolean;

Description Call *DispatchPopup* to simulate a click on the menu item associated with the pop-up menu handle. If *DispatchPopup* locates a menu item that matches the pop-up menu handle, it calls the *Click* method for that menu item and returns *True*. If there is no such item, *DispatchPopup* returns *False*.

See also *DispatchCommand* method, *IsShortCut* method, *Items* property, *TMenuItem.Click* method

FindItem

FindItem locates and returns a menu item.

function FindItem(Value: Word; Kind: TFindItemKind): TMenuItem;

Description Call *FindItem* to locate a menu item in the menu or one of its submenus. The menu item is identified by either a pop-up menu handle, command ID, or menu shortcut value. The *Kind* parameter specifies how the menu item is to be identified. It can be any of these values:

Value	Meaning
fkCommand	Command ID used by Windows WM_COMMAND message matches the *Value* parameter.

Value	Meaning
fkHandle	Pop-up menu handle matches the *Value* parameter.
fkShortCut	Menu shortcut key code matches the *Value* parameter.

See also *TMenuItem* object, *TMenuItem.Command* property, *TMenuItem.Handle* property, *TMenuItem.ShortCut* property

GetHelpContext

GetHelpContext returns the help context ID associated with a menu item.

```
function GetHelpContext(Value: Word; ByCommand: Boolean): THelpContext;
```

Description Call *GetHelpContext* to obtain the ID for context-sensitive help on a particular menu item. Identify the menu item as the item that matches the *Value* property. If *ByCommand* is *True*, *Value* is the *Command* property of the *TMenuItem*. If *ByCommand* is *False*, *Value* is the *Handle* property.

See also *FindItem* method, *TMenuItem.Command* property, *TMenuItem.Handle* property, *TMenuItem.HelpContext* property, *TWinControl.HelpContext* property

M

IsShortCut

IsShortCut dispatches a key message if it represents a shortcut key for an item in the menu.

```
function IsShortCut(var Message: TWMKey): Boolean;
```

Description Call *IsShortCut* when processing a keydown message to identify shortcut keys and trigger the appropriate menu action. *IsShortCut* decodes the key message passed in the *Message* parameter. It then encodes the state of the *Shift*, *Control*, and *Alt* keys with the value of the character that was typed to obtain the *ShortCut* property of a menu item. If *IsShortCut* locates a menu item associated with the shortcut key combination, it calls the item's *Click* method and returns *True*. If no item in the menu has a shortcut corresponding to the message, *IsShortCut* returns *False*.

See also *DispatchCommand* method, *DispatchPopup* method, *FindItem* method, *TMenuItem.Click* method

TMenuItem

Unit: Menus

TObject • TPersistent • TComponent • TMenuItem

TMenuItem describes the properties of an item in a menu.

Description Use *TMenuItem* to specify the appearance and behavior of an item in a menu. Each *TMainMenu* or *TPopupMenu* component can contain multiple menu items. When a menu is defined at design time, the Menu Designer automatically creates menu item objects for each command on the menu.

See also *TMainMenu* object, *TPopupMenu* object

Properties

The following table lists the properties of *TMenuItem*:

Property	Visibility	Default	Documented in
Break	published	mbNone	TMenuItem
Caption	published		TMenuItem
Checked	published	False	TMenuItem
Command	public, read-only		TMenuItem
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Count	public, read-only		TMenuItem
Default	published	False	TMenuItem
DesignInfo	public		TComponent
Enabled	published	True	TMenuItem
GroupIndex	published	0	TMenuItem
Handle	public, read-only		TMenuItem
HelpContext	published	0	TMenuItem
Hint	published		TMenuItem
Items	public, read-only		TMenuItem
MenuIndex	public		TMenuItem
Name	published		TComponent
Owner	public, read-only		TComponent
Parent	public, read-only		TMenuItem
RadioItem	published	False	TMenuItem
ShortCut	published	0	TMenuItem
Tag	published	0	TComponent
VCLComObject	public		TComponent
Visible	published	True	TMenuItem

Break

Break determines whether the menu item starts a new column in the menu.

property Break: TMenuBreak;

Description Set *Break* to break a long menu into columns at the item described by this menu item. These are the possible values:

Value	Meaning
mbNone	No menu breaking occurs. This is the default value.
mbBarBreak	The *Parent* menu breaks into another column with the menu item appearing at the top of the new column. A bar separates the new and the old columns.
mbBreak	The *Parent* menu breaks into another column with the menu item appearing at the top of the new column. Only space separates the new and the old columns.

See also *Caption* property, *Parent* property

Caption

Caption specifies the text of the menu item.

property Caption: **string**;

Description Set *Caption* to the string the user sees for this item in the menu. To underline a character in a string, include an ampersand (&) before the character. This type of character is called an accelerator character. If *Caption* includes an accelerator character, the user can select the menu item by pressing *Alt* while typing the underlined character.

Use the *Caption* property to identify the menu item as a line that separates the menu into parts. Specify a hyphen character (-) as the value of *Caption* for the menu item to indicate that the menu item is a separator. To separate the menu into multiple columns, use the *Break* property instead.

See also *Break* property, *Checked* property

Checked

Checked specifies whether a check mark should appear beside the *Caption*.

property Checked: Boolean;

Description Use *Checked* to allow a menu item to work like a check box. If *Checked* is *True*, the menu item appears checked. If *Checked* is *False*, the menu item remains unchecked.

See also *Caption* property, *RadioItem* property

Command

Command is the Windows command ID associated with the menu item.

property Command: Word;

Description Use *Command* in an application that directly handles *WM_COMMAND* messages. When the user selects the menu item, Windows sends a WM_COMMAND to the window associated with the menu. The *ItemID* of the message contains the value of the *Command* property of the menu item that was selected.

For example, the following procedure is a *WM_COMMAND* message handler. It checks the *ItemID* field of *Msg* to see if the message was generated by a menu item called *MenuThink*. If so, it displays a message dialog. When writing message handlers, remember to call Inherited afterward, if necessary, so Windows can perform default message processing.

```
procedure TForm1.WMCommand(var Msg: TWMCommand);
begin
  if Msg.ItemID = MenuThink.Command then
    MessageDlg('This is the Think command', mtInformation, [mbOk], 0);
  Inherited;
end;
```

See also *OnClick* event, *TMenu.DispatchCommand* method, *TMenu.WindowHandle* property

Count

Count is the number of subitems of the menu item.

property Count: Integer;

Description Read *Count* to determine the number of subitems listed in the *Items* property array. When the user clicks on a menu item that has subitems, a dropdown menu appears which displays those subitems. Each subitem can, in turn, contain additional subitems. The *Count* property counts only the immediate subitems of the menu item.

See also *Items* property, *MenuIndex* property

Default

Default specifies whether the menu item is invoked when the parent item is double clicked.

property Default: Boolean;

Description Set *Default* to specify whether a menu item is the default item in a submenu. *Default* menu items are executed when the parent item is double-clicked, allowing users to avoid navigating the submenus that contain them.

The default item appears in boldface. A submenu can have only one default item. If the *Items* list of the parent menu item already contains a default menu item, setting the *Default* property to *True* sets the *Default* property of the previous default item to *False*.

Note The *Default* property works under Windows NT 3.51 only if the "new shell" has been installed.

See also *Click* method, *Parent* property

Enabled

Enabled specifies whether the menu item is enabled.

property Enabled: Boolean;

Description Use *Enabled* to enable or disable a menu item. If *Enabled* is *True*, the *Click* method is called when the user selects the item with the mouse. If *Enabled* is *False*, the menu item appears dimmed and the user cannot select it.

Note Calling the *Click* method works even when *Enabled* is *False*.

See also *Click* method, *Visible* property

GroupIndex

GroupIndex identifies the logical group to which the menu item belongs.

property GroupIndex: Byte;

Description Use *GroupIndex* to control how menus are merged. In applications that contain multiple forms, items from the menus of secondary forms can be merged into the application's main menu as those forms become active. MDI applications always merge the menus of child windows with the main menu of the parent window.

By default, all menu items in a menu bar (main menu) have the same *GroupIndex* value. To use *GroupIndex* to merge menus, set the *GroupIndex* value of each successive menu item to a value equal to or greater than that of the previous menu item.

When a menu item in a menu bar that is merged into the application's main menu has the same *GroupIndex* value as a menu item in the main menu, the menu item in the application's main menu is replaced by the corresponding menu item. If multiple menu items in the application's main menu have the same *GroupIndex* value, they can be replaced only by separate menu items from the other form's menu that also have repeated *GroupIndex* values. Thus, the first main menu item with a given value is replaced by the first item in the second form with that value, the second item in the main form is replaced by the second item in the second form, and so on.

When a menu item in a menu bar that is merged into the application's main menu has a *GroupIndex* that falls between the *GroupIndex* values of two adjacent items on the application's main menu, that menu item is inserted between the two adjacent items. If the item's *GroupIndex* value is greater than all other *GroupIndex* values in the application's main menu, the item appears at the end of the main menu. If the *GroupIndex* value is less than all other *GroupIndex* values in the application's main menu, the menu item appears at the beginning of the main menu bar.

When an object created by an OLE 2.0 server application is activated, the server can try to merge its menus with the menus of the container application. The *GroupIndex* property is used to replace up to three main menu items with items from the server, in the manner described above. The server application uses the following predefined *GroupIndex* values to locate menu items to replace:

Group	Index	Description
Edit	1	Server menu items for editing the active OLE object
View	3	Server menu items for modifying the view of the OLE object.
Help	5	Server menu items for accessing the server's online Help.

Note See the documentation for the OLE server for information about whether it attempts menu merge during in-place activation.

GroupIndex can also be used to define groups of menu items that act like radio buttons. To allow menu items to behave like radio buttons, set the *RadioItem* property to all menu items in the group to *True*, and assign each menu item the same value of *GroupIndex*. Checking any menu item in the group will then uncheck the others.

See also *MenuIndex* property, *RadioItem* property, *TMainMenu.AutoMerge* property, *TMainMenu.Merge* method, *TMainMenu.PopulateOLE2Menu* method

Handle

Handle is the Windows menu handle of the drop-down menu associated with the menu item.

```
property Handle: HMENU;
```

Description Use *Handle* when calling a Windows API function that requires a menu handle. *Handle* is only valid if the *Count* property is greater than 0.

See also *Count* property, *Items* property

HelpContext

HelpContext is the help context ID associated with the menu item.

```
property HelpContext: THelpContext;
```

Description Use *HelpContext* to associate the menu item with a screen in the Help system of the application. Each screen in the Help system should have a unique context ID. When a menu item is selected using the keyboard, pressing *F1* displays a Help screen. Which Help screen appears depends on the value of the *HelpContext* property.

See also *TApplication.HelpFile* property, *TWinControl.HelpContext* property

Hint

Hint specifies the text string that can appear when the user moves the mouse pointer over a menu item.

property Hint: **string**;

Description Set *Hint* to a string that provides more information about the meaning of the menu item than the *Caption*. The hint text appears in a Help Hint window when the user pauses with the mouse over the menu item if Help Hints are enabled. It is also available for the code in the application's *OnHint* event handler.

The value of *Hint* can specify both a short value for the Help Hint window and a longer string to be used by the *OnHint* event handler. To provide both a short and a long hint, set *Hint* to the short string, followed by a vertical bar (|), followed by the long string.

See also *TApplication.Hint* property, *TApplication.OnHint* event, *TControl.Hint* property

Items

Items is an indexed array of menu items in the submenu of the menu item.

property Items[Index: Integer]: TMenuItem; **default**;

Description Use *Items* to access to a subitem by its position in the list of subitems. The value of *Index* is the position of the subitem within the *Items* array. For example, if an application has a File drop-down menu that contains the menu items New, Open, and Save, in that order, FileMenu.Items[2] refers to the Save command.

See also *Count* property, *MenuIndex* property, *Parent* property

MenuIndex

MenuIndex is index of the menu item within its parent menu.

property MenuIndex: Integer;

Description Use *MenuIndex* to locate the menu item in the *Items* property of its parent menu item. Changing this value moves the menu item within its menu.

Note *MenuIndex* may not correspond to the position of the item in the menu the user sees if the menu includes some items that are not visible.

See also *Count* property, *GroupIndex* property, *Items* property, *Parent* property, *Visible* property

Parent

Parent identifies the parent menu item of this menu item.

property Parent: TMenuItem;

Description Read *Parent* to determine the menu item that represents the dropdown menu that contains this menu item. If the menu item is a top level menu item in a main menu or pop-up menu, *Parent* is the value of that menu's *Items* property.

See also *Items* property, *MenuIndex* property, *TMenu.Items* property

RadioItem

RadioItem specifies whether the menu item is mutually exclusive with other menu items in its group.

property RadioItem: Boolean;

Description Use *RadioItem* to make the menu item behave like a radio button with a group of other menu items. When *RadioItem* is *True*, only one menu item in the group that contains the menu item can be selected at a time. The group of menu items is the set of all menu items in a pop-up or drop-down menu that have the same value of *GroupIndex*. The single selected menu item in the group is indicated by a round dot next to the *Caption*.

RadioItem does not work for the top-level items in a main menu.

Note Under Windows NT 3.51, the round dot indicator only appears if the "new shell" has been installed.

See also *Caption* property, *Checked* property, *GroupIndex* property, *TMainMenu* object

ShortCut

ShortCut specifies the key combination users can type to quickly access the menu item.

property ShortCut: TShortCut

Description Use *ShortCut* to allow the user to type a key combination instead of selecting the item using the menu. This key combination appears to the right of the menu item in the menu.

When setting menu shortcuts at design time, the Object Inspector provides a long list of key combinations to choose from. To create menu shortcuts at runtime, use the global *ShortCut* or *TextToShortCut* function. For example, the following line creates a shortcut, *Ctrl+C* and assigns it to a menu item named CloseCommand.

```
CloseCommand.ShortCut := ShortCut(Word('C'), [ssCtrl]);
```

See also *ShortCut* function, *ShortCutToKey* procedure, *ShortCutToText* function, *TextToShortCut* function, *TMenu.IsShortCut* method

Visible

Visible specifies whether the menu item appears in the menu.

```
property Visible: Boolean;
```

Description Use *Visible* to prevent the user from seeing an item in the menu. For example, remove menu items that do not apply to the current component of a pop-up menu in an *OnPopup* event handler.

If *Visible* is *True*, the menu item appears. If *Visible* is *False*, the menu item is hidden.

See also *Enabled* property, *TPopupMenu.OnPopup* event, *TPopupMenu.PopupComponent* property

Events

The following table lists the events of *TMenuItem*:

Event	Visibility	Documented in
OnClick	published	TMenuItem

OnClick

OnClick occurs when the user clicks menu item.

```
type TNotifyEvent = procedure (Sender: TObject) of object;
property OnClick: TNotifyEvent;
```

Description Write an *OnClick* event handler to implement the desired behavior for when the user selects the menu item. An advantage of using the *OnClick* event handler over directly handling WM_COMMAND messages sent when the user selects the menu item, is that only the *OnClick* event occurs when the *Click* method is called.

See also *Click* method, *Command* property

Methods

The following table lists the methods of *TMenuItem*:

Method	Visibility	Documented in
Add	public	TMenuItem
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject

Method	Visibility	Documented in
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Click	public	TMenuItem
Create	public	TMenuItem
DefaultHandler	public	TObject
Delete	public	TMenuItem
Destroy	public	TMenuItem
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
IndexOf	public	TMenuItem
InheritsFrom	public	TObject
InitInstance	public	TObject
Insert	public	TMenuItem
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Remove	public	TMenuItem
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Add

Add adds a menu item to the end of the *Items* property array.

```
procedure Add(Item: TMenuItem);
```

Description Use *Add* to add a new menu item to the dropdown menu for this menu item. If this menu item is the value of the *Items* property of a main menu or pop-up menu, *Add* adds an item to the end of the menu. Specify the new menu item to add as the value of the *Item* parameter.

For example, this code adds a menu item to a File menu:

```
procedure Form1.Button1Click(Sender: TSender);
var
  NewItem: TMenuItem;
begin
  NewItem := TMenuItem.Create(Self);
  NewItem.Caption := 'New item';
  File.Add(NewItem);
end;
```

See also *Delete* method, *Insert* method, *Items* property, *Remove* method,*TMenu.Items* property

Click

Click simulates a mouse click.

procedure Click; **virtual**;

Description *Click* generates an *OnClick* event, as if the user had clicked the menu item. Unlike a real mouse click, *Click* does not generate a WM_COMMAND message to the window associated with the menu.

Component writers can override click to provide a different response when the user selects the menu item.

See also *Command* property, *OnClick* event

Create

Create creates an instance of *TMenuItem*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to create a menu item at runtime. Menu items defined at design time using the Menu designer are created automatically. After calling the inherited constructor, *Create* initializes the *Visible* and *Enabled* properties to *True*, and generates a unique value for the *Command* property.

See also *Command* property, *Enabled* property, *Visible* property

Delete

Delete removes a menu item form the *Items* property array.

Procedure Delete(Index: Integer);

Description Call *Delete* to remove a menu item and all its submenus from a menu. The *Index* parameter is the index of the item to be deleted, where 0 specifies the first item in the menu, 1 specifies the second item, and so on.

To temporarily hide a menu item, without deleting it, use the *Visible* property.

See also *Add* method, *Insert* method, *Items* property, *Remove* method, *Visible* property

Destroy

Destroy destroys an instance of *TMenuItem*.

```
destructor Destroy; override;
```

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the menu item is not already freed, and only then calls *Destroy*.

See also *Free* method

IndexOf

IndexOf returns the position of a menu item within the *Items* array.

```
function IndexOf(Item: TMenuItem): Integer;
```

Description Call *IndexOf* to locate a menu item in the submenu headed by this menu item. *IndexOf* returns 0 for the first menu item, 1 for the second menu item, and so on. If the menu item is not in the submenu, *IndexOf* returns –1.

Note *IndexOf* may not correspond to the position of the item in the menu the user sees if the menu includes some items that are not visible.

See also *Items* property, *MenuIndex* property, *Visible* property

Insert

Insert inserts a menu item into a specified position in the *Items* array.

```
procedure Insert(Index: Integer; Item: TMenuItem);
```

Description Use *Insert* to insert a new menu item to the dropdown menu for this menu item. If this menu item is the value of the *Items* property of a main menu or pop-up menu, *Insert* inserts an item into that menu. Specify the new menu item to add as the value of the *Item* parameter, and the position in the *Items* array as the value of the *Index* parameter.

For example, this code inserts a menu item after the first item in a menu named FileMenu:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  NewItem: TMenuItem;
begin
  NewItem := TMenuItem.Create(FileMenu);
```

```
    try
      NewItem.Caption := 'Do this';
      FileMenu.Insert(1, NewItem);
    except
      NewItem.Free;
      raise; { reraise the exception }
    end;
  end;
```

See also *Add* method, *Delete* method, *Insert* method, *Items* property, *Remove* method, *TMenu.Items* property

Remove

Remove removes a menu item from the *Items* property array

procedure Remove(Item: TMenuItem);

Description Call *Remove* to remove a menu item and all its submenus from a menu. The *Item* parameter is the item to be removed. If *Item* is not a menu item in the *Items* array, *Remove* raises an exception.

To temporarily hide a menu item, without deleting it, use the *Visible* property.

See also *Add* method, *Delete* method, *EMenuError* object, *Insert* method, *Items* property, *Remove* method, *Visible* property

N

TNumericField

Unit: DB

TObject • TPersistent • TComponent • TField • TNumericField

TNumericField component is the common ancestor of all the numeric fields in a dataset.

Description *TNumericField* is an abstract base class that encapsulates the fundamental behavior common to all numeric field components. In addition to the properties, events, and methods common to all field components introduced by *TField*, *TNumericField* introduces new properties for formatting numeric data for display or editing purposes.

See also *TBCDField* component, *TFloatField* component, *TIntegerField* component

Properties

The following table lists the properties of *TNumericField*:

Property	Visibility	Default	Documented in
Alignment	published	taRightJustify	TField
AsBoolean	public		TField
AsCurrency	public		TField
AsDateTime	public		TField
AsFloat	public		TField
AsInteger	public		TField
AsString	public		TField
AsVariant	public		TField
AttributeSet	public		TField
Calculated	public	False	TField
CanModify	public, read-only		TField
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ConstraintErrorMessage	published		TField
CurValue	public, read-only		TField
CustomConstraint	published		TField
DataSet	public		TField
DataSize	public, read-only		TField
DataType	public, read-only		TField
DefaultExpression	published		TField
DesignInfo	public		TComponent
DisplayFormat	published		TNumericField
DisplayLabel	published		TField
DisplayName	public, read-only		TField
DisplayText	public, read-only		TField
DisplayWidth	published		TField
EditFormat	published		TNumericField
EditMask	public		TField
EditMaskPtr	public, read-only		TField
FieldKind	published	fkData	TField
FieldName	published		TField
FieldNo	public, read-only		TField
ImportedConstraint	published		TField
Index	published		TField
IsIndexField	public, read-only		TField
IsNull	public, read-only		TField
KeyFields	published		TField
Lookup	public		TField

Property	Visibility	Default	Documented in
LookupCache	published	False	TField
LookupDataSet	published		TField
LookupKeyFields	published		TField
LookupList	public, read-only		TField
LookupResultField	published		TField
Name	published		TComponent
NewValue	public		TField
Offset	public, read-only		TField
OldValue	public, read-only		TField
Origin	published		TField
Owner	public, read-only		TComponent
ReadOnly	published	False	TField
Required	published	False	TField
Size	public		TField
Tag	published	0	TComponent
Text	public		TField
ValidChars	public		TField
Value	public		TField
VCLComObject	public		TComponent
Visible	published	True	TField

DisplayFormat

DisplayFormat determines how a numeric field's value is formatted for display in a data-aware control.

```
property DisplayFormat: string;
```

Description Use the *DisplayFormat* property to override automatic formatting of a field for display purposes.

The value of *DisplayFormat* is a string that encodes the formatting of numeric data using the specifiers in the following table:

Specifier	Represents
0	Digit placeholder. If the value being formatted has a digit in the position where the "0" appears in the format string, then that digit is copied to the output string. Otherwise, a "0" is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the "#" appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.

Specifier	Represents
.	Decimal point. The first "." character in the format string determines the location of the decimal separator in the formatted value; any additional "." characters are ignored. The actual character used as a the decimal separator in the output string is determined by the *DecimalSeparator* global variable. The default value of *DecimalSeparator* is specified in the Number Format of the Regional Settings section in the Windows Control Panel.
,	Thousand separator. If the format string contains one or more "," characters, the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of "," characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as a the thousand separator in the output is determined by the *ThousandSeparator* global variable. The default value of *ThousandSeparator* is specified in the Number Format of the Regional Settings section in the Windows Control Panel.
E+	Scientific notation. If any of the strings "E+", "E–", "e+", or "e–" are contained in the format string, the number is formatted using scientific notation. A group of up to four "0" characters can immediately follow the "E+", "E–", "e+", or "e–" to determine the minimum number of digits in the exponent. The "E+" and "e+" formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The "E–" and "e–" formats output a sign character only for negative exponents.
'xx' / "xx"	Characters enclosed in single or double quotes are output as-is, and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost "0" before the decimal point in the format string and the rightmost "0" after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ("0" or "#") to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the "." character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain between one and three sections separated by semicolons.

- One section: The format string applies to all values.
- Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.
- Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If the section for negative values or the section for zero values is empty, that is, if there is nothing between the semicolons that delimit the section, the section for positive values is used instead.

If the section for positive values is empty, or if the entire format string is empty, the value is formatted using general floating-point formatting with 15 significant digits. General floating-point formatting is also used if the value has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

Note *DisplayFormat* affects only the display of the field's data. It does not affect the validity of strings assigned to a numeric field using the *AsString* property.

See also *AsString* property, *DisplayText* property, *EditFormat* property, *OnGetText* event

EditFormat

EditFormat determines how a numeric field's value is formatted when it is being edited in a data-aware control.

property EditFormat: **string**;

Description Use the *EditFormat* property to override the formatting of a field when the field's value is being edited in a data-aware control.

When *EditFormat* is unassigned, but the *DisplayFormat* property has a value, the *DisplayFormat* string is used.

The value of *EditFormat* is a string that encodes the formatting of numeric data using the specifiers in the following table:

Specifier	Represents
0	Digit placeholder. If the value being formatted has a digit in the position where the "0" appears in the format string, then that digit is copied to the output string. Otherwise, a "0" is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the "#" appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.
.	Decimal point. The first "." character in the format string determines the location of the decimal separator in the formatted value; any additional "." characters are ignored. The actual character used as a the decimal separator in the output string is determined by the *DecimalSeparator* global variable. The default value of *DecimalSeparator* is specified in the Number Format of the Regional Settings section in the Windows Control Panel.

N

Specifier	Represents
,	Thousand separator. If the format string contains one or more "," characters, the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of "," characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as a the thousand separator in the output is determined by the *ThousandSeparator* global variable. The default value of *ThousandSeparator* is specified in the Number Format of the Regional Settings section in the Windows Control Panel.
E+	Scientific notation. If any of the strings "E+", "E–", "e+", or "e–" are contained in the format string, the number is formatted using scientific notation. A group of up to four "0" characters can immediately follow the "E+", "E–", "e+", or "e–" to determine the minimum number of digits in the exponent. The "E+" and "e+" formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The "E–" and "e–" formats output a sign character only for negative exponents.
'xx'/ "xx"	Characters enclosed in single or double quotes are output as-is, and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost "0" before the decimal point in the format string and the rightmost "0" after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ("0" or "#") to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the "." character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain between one and three sections separated by semicolons.

- One section: The format string applies to all values.
- Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.
- Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If the section for negative values or the section for zero values is empty, that is if there is nothing between the semicolons that delimit the section, the section for positive values is used instead.

If the section for positive values is empty, or if the entire format string is empty, the value is formatted using general floating-point formatting with 15 significant digits. General floating-point formatting is also used if the value

has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

Note *EditFormat* affects only the display of the field's data when it is being edited. It does not affect the validity of strings assigned to a numeric field using the *AsString* property, nor does it affect the display of the field's data when it is not being edited.

See also *DisplayFormat* property, *OnGetText* event, *Text* property

Events

The following table lists the events of *TNumericField*:

Event	Visibility	Documented in
OnChange	published	TField
OnGetText	published	TField
OnSetText	published	TField
OnValidate	published	TField

Methods

The following table lists the methods of *TNumericField*:

Method	Visibility	Documented in
Assign	public	TField
AssignValue	public	TField
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TField
Create	public	TNumericField
DefaultHandler	public	TObject
Destroy	public	TField
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
FocusControl	public	TField
Free	public	TObject

Method	Visibility	Documented in
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetData	public	TField
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
IsBlob	public	TField
IsValidChar	public	TField
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RefreshLookupList	public	TField
RemoveComponent	public	TComponent
SafeCallException	public	TComponent
SetData	public	TField
SetFieldType	public	TField
Validate	public	TField

Create

The Create method creates an instance of a *TNumericField* object.

constructor Create(AOwner: TComponent); **override**;

Description Do not call the create method for *TNumericField* directly. *TNumericField* is an abstract class and its constructor should only be called as an inherited method from the constructor of a derived class.

After calling the inherited *Create* method, *Create* changes the *Alignment* property to *taRightJustify*.

See also *Alignment* property, *TField.Create* method

TObject

Unit: System

TObject

TObject is the ultimate ancestor for all other objects and components in Delphi.

Description *TObject* encapsulates the fundamental behavior common to all objects in Delphi by introducing interfaces for methods that provide:

- The ability to create, maintain and destroy an instance of the object by allocating, initializing and freeing the memory needed for that object.
- Class type and instance information on an object, and runtime type information (RTTI) about its published properties.
- Support for message-handling.

Use *TObject* as an immediate base class when declaring simple objects that are not components, and that do not need streaming or assignment capabilities. If no ancestor type is specified when declaring a new object type, Delphi automatically uses *TObject* as the ancestor.

Much of the powerful capability of Delphi objects is established by the methods that *TObject* introduces. Many of these methods are used internally by the Delphi environment and are not intended for users to call directly. Several others are generally overridden in descendant objects and components that have more complex behavior.

Note While *TObject* is not technically an abstract class, objects of this type are not normally instantiated.

See also *TComponent* object, *TPersistent* object

Methods

The following table lists the methods of *TObject*:

Method	Visibility	Documented in
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TObject
DefaultHandler	public	TObject
Destroy	public	TObject

Method	Visibility	Documented in
Dispatch	public	TObject
FieldAddress	public	TObject
Free	public	TObject
FreeInstance	public	TObject
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
InheritsFrom	public	TObject
InitInstance	public	TObject
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
SafeCallException	public	TObject

ClassInfo

ClassInfo returns a pointer to the runtime type information (RTTI) table for the object type.

```
class function ClassInfo: Pointer;
```

Description Use *ClassInfo* to access the RTTI table that contains information about the object type, its ancestor type, and all of its published properties.

RTTI is used extensively for internal use in the Delphi environment. *ClassInfo* is rarely called directly in an application. *TObject* provides other methods specifically intended to provide easier access to RTTI information.

See also *ClassType* method

ClassName

ClassName returns a string containing the name of the actual type of the object instance that is referenced by a variable, rather than the type of the variable.

```
class function ClassName: ShortString;
```

Description Use *ClassName* to find the actual type of an object in the format of a string, such as when passing a string as a parameter to another routine. For example, a variable of an ancestor type can refer to an instance of any descendant type. In this case, *ClassName* returns a string indicating the actual type of the object, rather than a string containing the type of the declared variable.

ClassName provides information on objects across modules so it is especially useful for exception-handling.

When reporting exception-handling errors on a particular object *ClassName* is used by the **on** statement does a string comparison of classes.

See also *ClassNameIs* method, *ClassParent* method, *ClassType* method

ClassNameIs

ClassNameIs determines whether an object is of a specific type.

```
class function ClassNameIs(const Name: string): Boolean;
```

Description Use *ClassNameIs* when writing conditional code based upon an object's type or to query objects across modules, or DLLs.

ClassNameIs returns *True* if the string passed in the *Name* parameter matches the name of the class. Otherwise, it returns *False*.

See also *ClassName* property

ClassParent

ClassParent returns the type of the immediate ancestor of a class.

```
class function ClassParent: TClass;
```

Description *ClassParent* returns **nil** for type *TObject* because *TObject* has no parent.

ClassParent is used internally in Delphi for RTTI, by the **is** and **as** operators, and by the *InheritsFrom* method. Because it is a class method, it can be used to determine the type of an class reference.

Note Applications should rarely, if ever, call *ClassParent* directly.

See also *ClassInfo* method, *ClassType* method, *InheritsFrom* method

ClassType

ClassType returns a pointer to the runtime type information of an object instance.

```
function ClassType: TClass;
```

Description *ClassType* dynamically determines the actual type of an object. *ClassType* is used internally in Delphi by the **is** and **as** operators. Do not use *ClassType* directly.

See also *ClassInfo* method, *ClassName* method, *ClassParent* method

CleanupInstance

CleanupInstance performs finalization on long strings, variants, and interface variables within a class.

```
procedure CleanupInstance;
```

Description Do not call *CleanupInstance* directly. *CleanupInstance* is called automatically when *Free* is called.

CleanupInstance releases all long strings and variants. It set long strings to empty and variants to *Unassigned*.

See also *Destroy* method, *Free* method

Create

Create is constructs an object and initialize its data before the object is first used.

```
constructor Create;
```

Description *Create* constructs a properly initialized instance of an object. The purpose, size and behavior of the objects in Delphi can differ greatly. The *Create* method defined by *TObject* does nothing special; that is, it does not initialize any data. Memory allocation is, however, handled automatically when *Create* is called.

Note Descendant objects usually define a constructor that is customized to create that particular kind of object initializing its data as needed.

If an exception escapes from a constructor, the object's destructor will be called to clean up the failed instance.

See also *Destroy* method, *Free* method

DefaultHandler

DefaultHandler introduces an interface for processing a message record passed to it.

```
procedure DefaultHandler(var Message); virtual;
```

Description *DefaultHandler* is called by *Dispatch* when it cannot find a method for a particular message. *DefaultHandler* provides message handling for all messages for which an object does not have specific handlers. Descendant classes of *TObject* that process messages usually define *DefaultHandler* methods according to the type of messages they handle. For example, *TWinControl* overrides *DefaultHandler* to call *DefWindowProc*.

Calling **inherited** in a message-handling method results in a call to the ancestor's *DefaultHandler* method only if that ancestor does not specify a message method for the particular message being handled. Otherwise calling inherited results in a call to the specific handler for that type of message.

See also *Dispatch* method

Destroy

Destroy is a destructor that disposes of an object instance.

```
destructor Destroy; virtual;
```

Description It is not recommended coding style to call *Destroy* directly; call *Free* instead. *Free* checks to ensure that the object instance is not **nil** before calling *Destroy*.

Destroy defined by *TObject* does nothing special, however memory deallocation is handled automatically when *Destroy* is called. Descendant objects usually define a destructor that is customized to destroy that particular kind of object.

When declaring a *Destroy* method in a descendant object type, always add the **override** directive to the declaration and call the inherited *Destroy* as the last statement in the overriding method. Since *Destroy* is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note If an exception escapes from the constructor, the destructor will be called to destroy the partially constructed object instance that failed to initialize completely.

Destroy should be implemented so that it calls *Free* on all sub-objects created within the object's constructor (that is, allocated by the constructor). Unlike *Destroy*, *Free* provides a safeguard when destroying objects that are **nil**. The destructor should also check that other allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

See also *Free* method

Dispatch

Dispatch calls message-handling methods for the object, based on the contents of the *Message* parameter.

```
procedure Dispatch(var Message);
```

Description Call *Dispatch* to automatically pass messages to the current event handler for an object.

Dispatch determines whether a message is in the list of message handlers declared for the object. If the object does not handle the message, *Dispatch* then examines the message-handler list of the ancestor type, and continues checking ancestors until it either finds a specific handler or runs out of ancestors, in which case it calls *DefaultHandler*.

The *Message* parameter is untyped for *Dispatches*. Although any kind of data can be passed to *Dispatch*, use a message record such as *TMessage* or a specific message-record type. The only assumption *Dispatch* makes about the data in *Message* is that the first two bytes contain a message ID, that is an integer-that determines which message handler *Dispatch* calls to handle the message.

See also *DefaultHandler* method, *TMessage* type

FieldAddress

FieldAddress returns the address of a published object field.

```
function FieldAddress(const Name: ShortString): Pointer;
```

Description *FieldAddress* is used internally by the Delphi streaming system to access a published specified field of an object. *FieldAddress* returns a pointer to the field if it exists. If the object has no published field by that name, *FieldAddress* returns **nil**.

Users should access and manipulate fields by using properties instead of *FieldAddress*.

Free

Free destroys an object and frees its associated memory, if necessary.

```
procedure Free;
```

Description Use *Free* to destroy an object. *Free* automatically calls the *Destroy* if the object instance is not **nil.** Any object instantiated by a call to *Create* should be destroyed by a call to *Free*, so that the object an be properly destroyed and the memory released. *Free* is successful even if the object is **nil**, so if the object was never initialized, for example, calling *Free* won't result in an error.

See also *Destroy* method, *TForm.Release* method

FreeInstance

FreeInstance deallocates memory allocated by a previous call to the *NewInstance* method.

```
procedure FreeInstance; virtual;
```

Description All destructors call *FreeInstance* automatically to deallocate memory that was allocated by overriding *NewInstance*.

Do not call *FreeInstance* directly. However, *FreeInstance* should be overridden if *NewInstance* was overridden to change the way the object's instance data was allocated.

Like *NewInstance*, *FreeInstance* uses the value returned from *InstanceSize* to deallocate the object's memory.

See also *InstanceSize* method, *NewInstance* method

GetInterface

GetInterface retrieves the interface identified by a given identifier.

```
function GetInterface(const IID: TGUID; out Obj): Boolean;
```

Description *GetInterface* is used internally by the **as** operator to retrieve the interface implemented by an object. *GetInterface* is used by the *TInterfacedObject* to implement the *QueryInterface* method. An interface type name, such as *IUnknown*, can be supplied as the IID parameter since the compiler will replace the interface with the associated GUID.

Use *GetInterface* to obtain an interface from a VCL object instance when the object may not support the interface, but an exception should not be raised upon failure. Unlike the **as** operator, which will raise an exception on failure, *GetInterface* will not.

GetInterface returns, as the **out** parameter *Obj*, a pointer to the interface identified by the *IID* parameter. If the interface is supported by the object, *GetInterface* returns *True*. If the interface is not supported by the object, *GetInterface* returns *False*, and the *Obj* parameter is **nil**.

GetInterfaceEntry

GetInterfaceEntry returns the entry for a specific interface implemented in a class.

```
type
  PInterfaceEntry = ^TInterfaceEntry;
  TInterfaceEntry = record
    IID: TGUID;
    VTable: Pointer;
    IOffset: Integer;
  end;

class function GetInterfaceEntry(const IID: TGUID): PInterfaceEntry;
```

Description *GetInterfaceEntry* returns the entry for the interface, specified by the IID parameter, in the class. *GetInterfaceEntry* is used internally. It is rarely called directly.

Note One use for this method is to describe to OLE how to automatically dispatch calls to a dual-*IDispatch* interface. An interface type name, such as *IUnknown*, can be supplied as the IID parameter since the compiler will replace the interface with the associated GUID.

GetInterfaceTable

GetInterfaceTable returns the pointer to a structure containing all of the interfaces implemented by a given class.

```
type
  PInterfaceTable = ^TInterfaceTable;
  TInterfaceTable = record
    EntryCount: Integer;
    Entries: array[0..9999] of TInterfaceEntry;
  end;
```

class function GetInterfaceTable: PInterfaceTable;

Description *GetInterfaceTable* returns the *TInterfaceEntry*s for the class. This list only contains interfaces implemented by this class, not its ancestor. To find the ancestor list, call *GetParentClass* and then call its *GetInterfaceTable* method. To find the entry for a specific interface, use the *GetInterfaceEntry* method instead.

See also *GetInterfaceEntry* method

InheritsFrom

InheritsFrom is a class method, it is used to determine the relationship of two object types.

class function InheritsFrom(AClass: TClass): Boolean;

Description Use *InheritsFrom* to determine if a particular class type is an ancestor of an object. *InheritsFrom* returns *True* if the object type specified in the AClass parameter is an ancestor of the object type or the type of the object itself. Otherwise, it returns *False*.

Note The **is** and **as** operators use *InheritsFrom* in their implementations. The **is** operator, however, can only determine the inheritance relationship of an instance. As a class method *InheritsFrom* can determine the relationship of class references.

See also *ClassParent* method

InitInstance

InitInstance initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

class procedure InitInstance(Instance: Pointer): TObject;

Description There should be no reason to call *InitInstance* directly. *InitInstance* is called automatically by *NewInstance* when an object is created. When overriding *NewInstance* be sure to call *InitInstance* as the last statement.

InitInstance is not virtual, so you cannot override it. Initialize any data for an object in the constructor. Override *NewInstance* only for special memory allocation requirements, such as allocating all instances of a class type from a common memory pool.

See also *Create* method, *NewInstance* method

InstanceSize

InstanceSize returns the size in bytes of each instance of the object type.

class function InstanceSize: Longint;

Description Use *InstanceSize* to determine how many bytes of memory are required for a class's instance data. Delphi uses *InstanceSize* internally for methods that allocate and deallocate memory. *InstanceSize* is not a virtual method, so it cannot be overridden. *InitInstance* should only be called in the context of implementing a custom version of *NewInstance*.

See also *FreeInstance* method, *NewInstance* method

MethodAddress

MethodAddress returns the address of a published method.

```
class function MethodAddress(const Name: ShortString): Pointer;
```

Description *MethodAddress* is used internally in Delphi by the streaming system. *MethodAddress* is used to convert a method name, specified by *Name*, to a pointer containing the method address when reading in from a stream. There should be no need to call *MethodAddress* directly.

If *Name* does not specify a published method for the object, *MethodAddress* returns **nil**.

See also *MethodName* method

MethodName

MethodName returns a string containing the name of the method located at Address.

```
class function MethodName(Address: Pointer): ShortString;
```

Description *MethodName* is used internally in Delphi by the streaming system. *MethodName* is used to convert a pointer containing the method address to a string containing the method name when writing event properties to a stream. It is essentially the counterpart to *MethodAddress*. There should be no need to call *MethodName* directly.

If Address does not point to a published method of the object, *MethodName* returns an empty string.

See also *MethodAddress* method

NewInstance

NewInstance allocates memory for each instance of an object type and returns a pointer to that new instance.

```
class function NewInstance: TObject; virtual;
```

Description All constructors call *NewInstance* automatically. *NewInstance* calls *InstanceSize* to determine how much memory to allocate from the heap to contain a particular instance. Do not call *NewInstance* directly.

A descendant class can override *NewInstance* to change the way that particular class allocates its memory. Overriding *NewInstance* to change the memory allocation for an object type, might necessitate overriding *FreeInstance* to deallocate that same memory. For example, when allocating a large number of identical objects that all need to be in memory at the same time, one could allocate a large block of memory for the entire group, then override *NewInstance* to use part of that larger block for each instance. When overriding *NewInstance* call *InitInstance*, and not *inherited*, as the last statement.

Note By default *NewInstance* calls *InitInstance*.

See also *FreeInstance* method, *InstanceSize* method

SafeCallException

SafeCallException is an error handling method used to support OLE exceptions.

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): Integer; virtual;
```

Description *SafeCallException* is used for OLE exception handling support. For *TObject SafeCallException* returns E_UNEXPECTED. This is a default case that is appropriate for classes that do not support any interfaces. Classes that implement COM and OLE interfaces should override this method to handle other errors that might occur. COM and OLE classes in Delphi override *SafeCallException*.

TOleContainer

Unit: OleCtnrs | Palette: System

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomControl • TOleContainer

TOleContainer embeds or links OLE objects.

Description Use *TOleContainer* to handle many of the complexities of OLE 2.0. *TOleContainer* lets the user choose an OLE object to insert by simply calling the *InsertObjectDialog* method. *TOleContainer* can create either an embedded OLE object or a linked OLE object.

TOleContainer automatically handles menu merging—the process of combining the container form's menu with that of an in-place activated OLE object's server application. The menu items' *GroupIndex* property settings control how menus are merged. Those main menu items with *GroupIndex* values of 0, 2, and 4 remain; *TOleContainer* merges the server application's menus and replaces the main menu items with GroupIndex values of 1, 3, and 5 (if they exist).

OLE objects that are activated in-place add their servers' toolbars directly into the container application's window. Normally, any panels used for

toolbars are replaced by the OLE object's server's toolbars. Prevent this by setting a panel's *Locked* property to *True*.

When using *TOleContainer* in an SDI application (the main form's *FormStyle* property is *fsNormal* rather than *fsMDIForm*), place the *TOleContainer* component inside a panel whose *Align* property is set to *alClient*. *TOleContainer* replaces an SDI form's toolbars as described above, and using a panel automatically adjusts the amount of space available to the OLE container.

See also *TForm* object, *TMainMenu* object, *TMenuItem* object, *TOleControl* object, *TOleControlSite* object, *TPanel* object

Properties

The following table lists the properties of *TOleContainer*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
AllowActiveDoc	published	True	TOleContainer
AllowInPlace	published	True	TOleContainer
AutoActivate	published	aaDoubleClick	TOleContainer
AutoVerbMenu	published	True	TOleContainer
BorderStyle	published	bsSingle	TOleContainer
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
CanPaste	public, read-only		TOleContainer
Caption	published		TControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
CopyOnSave	published	True	TOleContainer
Ctl3D	published		TWinControl

Property	Visibility	Default	Documented in
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Iconic	published		TOleContainer
Left	published		TControl
Linked	public, read-only		TOleContainer
Modified	public		TOleContainer
Name	published		TControl
NewInserted	public, read-only		TOleContainer
ObjectVerbs	public, read-only		TOleContainer
OldStreamFormat	published	False	TOleContainer
OleClassName	public, read-only		TOleContainer
OleObject	public, read-only		TOleContainer
OleObjectInterface	public, read-only		TOleContainer
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
PrimaryVerb	public, read-only		TOleContainer
ShowHint	published		TControl
Showing	public, read-only		TWinControl
SizeMode	published	smClip	TOleContainer
SourceDoc	public, read-only		TOleContainer
State	public, read-only		TOleContainer
StorageInterface	public, read-only		TOleContainer
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

AllowActiveDoc

AllowActiveDoc specifies whether the OLE container supports the *IOleDocumentSite* interface.

```
property AllowActiveDoc: Boolean;
```

Description Set *AllowActiveDoc* to *True* to specify that the OLE container can house an ActiveDoc (DocObject). Set *AllowActiveDoc* to *False* to limit the OLE container to hosting OLE controls that use the more usual interfaces such as *IOleClientSite*. ActiveDoc-enabled controls have a slightly different border. Set *AllowActiveDoc* to *False* to maintain a consistent appearance in forms that do not need to support the *IOleDocumentSite* interface.

See also *AllowInPlace* property

AllowInPlace

AllowInPlace determines whether the OLE container allows inplace activation.

```
property AllowInPlace: Boolean;
```

Description Set *AllowInPlace* to specify what happens when the user activates the OLE object. If *AllowInPlace* is *True* and *Iconic* is *False*, the OLE object is activated inplace. If *AllowInPlace* is *False*, the OLE object is activated in a separate window. The default value is *True*.

See also *AllowActiveDoc* property, *AutoActivate* property, *Iconic* property

O

AutoActivate

AutoActivate determines how the object in an OLE container is activated.

```
type TAutoActivate = (aaManual, aaGetFocus, aaDoubleClick);

property AutoActivate: TAutoActivate;
```

Description Set *AutoActivate* to specify what actions activate the OLE object. *AutoActivate* takes one of the values in the following table:

Values	Meaning
aaManual	The OLE object must be activated programmatically by calling *DoVerb*(*ovShow*).
aaGetFocus	The OLE object is activated when the OLE container gets focus (by clicking on it with the mouse or pressing *Tab* to move the focus to it).
aaDoubleClick	(Default) The OLE object is activated by double-checking it or pressing *Enter* while the container has the focus.

See also *AllowActiveDoc* property, *DoVerb* method

AutoVerbMenu

AutoVerbMenu determines whether the OLE container automatically creates a pop-up menu containing the OLE object's verbs.

```
property AutoVerbMenu: Boolean;
```

Description Use *AutoVerbMenu* to specify whether the OLE object can replace the OLE container's popup menu. If *AutoVerbMenu* is *True* (the default), the OLE object replaces any existing *PopupMenu* property. If *AutoVerbMenu* is *False*, no popup menu is automatically created.

See also *DoVerb* method, *ObjectVerbs* property, *PopupMenu* property

BorderStyle

BorderStyle determines whether a single line is drawn around the OLE container.

```
type TBorderStyle = bsNone..bsSingle;
property BorderStyle: TBorderStyle;
```

Description Set *BorderStyle* to specify whether a the OLE container should be framed by a single line. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

See also *Ctl3D* property

CanPaste

CanPaste indicates whether the Windows clipboard contains an OLE object that can be pasted into the OLE container.

```
property CanPaste: Boolean;
```

Description Read *CanPaste* before trying to paste or paste link an OLE object from the clipboard.

See also *Paste* method, *PasteSpecialDialog* method

CopyOnSave

CopyOnSave determines whether the *SaveToFile* and *SaveToStream* methods first create a copy of the OLE object.

```
property CopyOnSave: Boolean;
```

Description When *CopyOnSave* is *True*, the *SaveToFile* and *SaveToStream* methods create a temporary OLE object, compressing redundant data to save space. This compressed copy is then saved to the file or stream. When *CopyOnSave* is *False*, the OLE object is saved directly to the file or stream without using a temporary copy to compress redundant information.

Set *CopyOnSave* to *False* when the container holds very large embedded objects and there might not be enough memory to make the temporary copy. The default value is *True*.

See also *OleObject* property, *SaveToFile* method, *SaveToStream* method

Iconic

Iconic determines whether the OLE object is displayed as an icon.

```
property Iconic: Boolean;
```

Description Set *Iconic* to specify whether the OLE container displays the icon for the server application instead of the image of the object. If *Iconic* is *True*, the OLE object is displayed as an icon; if *False*, it's displayed as it would be in the server application.

See also *GetIconMetaPict* method

Linked

Linked indicates whether the OLE object is linked or embedded.

```
property Linked: Boolean;
```

Description Read *Linked* to determine whether the OLE object is linked to the OLE server. If *Linked* is *True*, the OLE object is linked; if *False*, it's embedded. An OLE object must already be loaded in the container before accessing the *Linked* property.

See also *UpdateObject* method

Modified

Modified indicates whether the OLE object has been modified.

```
property Modified: Boolean;
```

Description Use *Modified* to determine when the OLE object has been modified (including being deleted or replaced by another OLE object). To keep track of incremental changes, set *Modified* to *False* and later check it to see if the OLE object has been modified after since *Modified* was set to *False*. If there is no OLE object loaded into the container, *Modified* returns *False*.

See also *OleObject* property

NewInserted

NewInserted indicates whether the OLE object was created rather than pasted in or created as a link to a file.

property NewInserted: Boolean;

Description Read *NewInserted* to determine whether the OLE object was newly created via a call to *InsertObjectDialog*. If there is no OLE object loaded into the container, *NewInserted* returns *False*. If *NewInserted* is *True* after a call to *InsertObjectDialog*, call *DoVerb*(*ovShow*) to let the user edit the object.

See also *DoVerb* method, *InsertObjectDialog* method, *OleObject* property

ObjectVerbs

ObjectVerbs is a list of the verbs the OLE object supports.

property ObjectVerbs: TStrings;

Description Read *ObjectVerbs* to determine the verbs that can be used with the OLE object. The verb names can have embedded ampersand (&) characters to indicate shortcut keys. An OLE object must already be loaded in the container before accessing the *ObjectVerbs* property.

See also *AutoVerbMenu* property, *DoVerb* method, *PrimaryVerb* property, *TStrings* object

OldStreamFormat

OldStreamFormat indicates whether the OLE object is loaded and saved using the Delphi 1.0 format.

property OldStreamFormat: Boolean;

Description Set *OldStreamFormat* to *True* to read OLE objects that were saved to file using Delphi 1.0 or to save OLE objects that will be used by Delphi 1.0. If *OldStreamFormat* is *False* (the default), the *LoadFromFile*, *LoadFromStream*, *SaveToFile*, and *SaveToStream* methods use the newer format that contains extra header information. The extra header information is used to validate a file or stream. This property is provided for backward compatibility only.

See also *LoadFromFile* method, *LoadFromStream* method, *SaveToFile* method, *SaveToStream* method

OleClassName

OleClassName is the class name of the OLE object.

property OleClassName: **string**;

Description Read *OleClassName* to obtain the class name (programmatic identifier) for the OLE object. The class name is a string describing the class of OLE object that

is registered in the Windows registry. An OLE object must already be loaded in the container before accessing the *OleClassName* property.

See also *OleObject* property

OleObject

OleObject is the OLE object hosted by the OLE container.

```
property OleObject: Variant;
```

Description Read *OleObject* to access the OLE object hosted by the OLE container as a variant. To interact with the OLE object using the *IOleObject* interface, use the *OleObjectInterface* property instead. If the OLE object is not loaded in the container before reading *OleObject*, an *EOleError* exception is raised.

See also *EOleError* object, *OleObjectInterface* property

OleObjectInterface

OleObjectInterface is the *IOleObject* interface for the OLE object.

```
property OleObjectInterface: IOleObject;
```

Description Use *OleObjectInterface* for low-level access to the OLE API. If there is no OLE object loaded into the container, *OleObjectInterface* returns **nil**.

See also *OleObject* property

O

PrimaryVerb

PrimaryVerb is the index of the primary (default) verb for the OLE object.

```
property PrimaryVerb: Integer;
```

Description Use *PrimaryVerb* to determine which verb in the *ObjectVerbs* list specifies the default action for the OLE object. In a call to the *DoVerb* method, *DoVerb*(*ovPrimary*) is the same as *DoVerb*(*PrimaryVerb*). An OLE object must already be loaded in the container before accessing the *PrimaryVerb* property.

See also *DoVerb* method, *ObjectVerbs* property

SizeMode

SizeMode determines how the OLE object is sized in the OLE container.

```
type TSizeMode = (smClip, smCenter, smScale, smStretch, smAutoSize);
property SizeMode: TSizeMode;
```

Description Set *SizeMode* to specify how the OLE object is fit into the client area of the OLE container. The following table gives the possible values:

Value	Meaning
smClip	(Default)- Displays the OLE object at its normal size, clipping any parts that don't fit within the container.
smCenter	Displays the OLE object at its normal size, centering it within the container.
smScale	Scales or shrinks the view of the OLE object to fit within the container, by scaling width and height proportionally.
smStretch	Scales or shrinks the view of the OLE object to fill the OLE container, without regard to preserving the proportions of the OLE object.
smAutoSize	Displays the OLE object at its normal size and automatically resizes the container to fit the size of the OLE object.

See also *AllowInPlace* property, *ClientRect* property

SourceDoc

SourceDoc is the name of the source document for a linked OLE object.

```
property SourceDoc: string;
```

Description Read *SourceDoc* to obtain the name of the OLE object when it is linked. An OLE object must already be loaded in the container before accessing the *SourceDoc* property. If the OLE object isn't linked, *SourceDoc* returns an empty string.

See also *Linked* property, *OleClassName* property

State

State provides information about the state of the OLE object.

```
type TObjectState = (osEmpty, osLoaded, osRunning, osOpen, osInPlaceActive, osUIActive);
property State: TObjectState;
```

Description Read *State* to determine the state of the OLE object and its server application. *State* has one of the following values:

Values	Meaning
osEmpty	There is no OLE object in the container.
osLoaded	There is an OLE object in the container, but its server application isn't currently running.
osRunning	The OLE object's server is running.
osOpen	The OLE object is open in a separate window.

Values	Meaning
osInPlaceActive	The OLE object is activated inplace, but hasn't yet merged its menus or toolbars. This is an intermediate state; the State property will shift to osUIActive as soon as the menus and toolbars have been merged.
osUIActive	The OLE object is activated inplace and menus and toolbars have been merged

See also *AllowInPlace* property, *Iconic* property

StorageInterface

StorageInterface is the *IStorage* interface for the OLE object.

```
property StorageInterface: IStorage;
```

Description Use *StorageInterface* for low-level access to the OLE API. If there is no OLE object loaded into the container, *StorageInterface* returns **nil**.

See also *OleObjectInterface* property

Events

The following table lists the events of *TOleContainer*:

Event	Visibility	Documented in
OnActivate	published	TOleContainer
OnDeactivate	published	TOleContainer
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnObjectMove	published	TOleContainer
OnResize	published	TOleContainer
OnStartDrag	published	TControl

OnActivate

OnActivate occurs immediately after the OLE object is activated.

```
type TNotifyEvent = procedure(Sender: TObject) of object;
property OnActivate: TNotifyEvent;
```

Description Write an *OnActivate* event handler to take specific action when the OLE object becomes active.

See also *AutoActivate* property, *DoVerb* method, *OnDeactivate* event, *State* property

OnDeactivate

OnDeactivate occurs immediately after the OLE object is deactivated.

```
type TNotifyEvent = procedure(Sender: TObject) of object;
property OnDeactivate: TNotifyEvent;
```

Description Write an *OnDeactivate* event handler to respond when the OLE object is deactivated.

See also *Close* method, *OnActivate* event, *State* property

OnObjectMove

OnObjectMove occurs when the OLE container is notified that the OLE object has moved or resized.

```
type TObjectMoveEvent = procedure(OleContainer: TOleContainer; const Bounds: TRect)
  of object;
property OnObjectMove: TObjectMoveEvent;
```

Description Write an *OnObjectMove* event handler to respond when the OLE object changes size or position. The *Bounds* parameter specifies the boundaries of the in-place object in the client coordinates of OLE container. *OnObjectMove* occurs before the OLE object's image is updated to reflect the change.

See also *OnResize* event

OnResize

OnResize occurs immediately after the OLE container window is resized.

```
type TNotifyEvent = procedure(Sender: TObject) of object;
property OnResize: TNotifyEvent;
```

Description Write an *OnResize* event handler to respond to changes in the size of the OLE container window. If *SizeMode* is *smAutoSize*, the container window changes size whenever the OLE object resizes.

See also *OnObjectMove* event, *SizeMode* property

Methods

The following table lists the methods of *TOleContainer*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ChangeIconDialog	public	TOleContainer
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
Close	public	TOleContainer
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Copy	public	TOleContainer
Create	public	TOleContainer
CreateLinkToFile	public	TOleContainer
CreateObject	public	TOleContainer
CreateObjectFromFile	public	TOleContainer
CreateObjectFromInfo	public	TOleContainer
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TOleContainer
DestroyComponents	public	TComponent
Destroying	public	TComponent
DestroyObject	public	TOleContainer
DisableAlign	public	TWinControl
Dispatch	public	TObject
DoVerb	public	TOleContainer
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl

Method	Visibility	Documented in
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetIconMetaPict	public	TOleContainer
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InsertObjectDialog	public	TOleContainer
InstanceSize	public	TObject
Invalidate	public	TWinControl
LoadFromFile	public	TOleContainer
LoadFromStream	public	TOleContainer
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
ObjectPropertiesDialog	public	TOleContainer
PaintTo	public	TWinControl
Paste	public	TOleContainer
PasteSpecialDialog	public	TOleContainer
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
Run	public	TOleContainer
SafeCallException	public	TComponent
SaveToFile	public	TOleContainer

Method	Visibility	Documented in
SaveToStream	public	TOleContainer
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl
UpdateObject	public	TOleContainer
UpdateVerbs	public	TOleContainer

ChangeIconDialog

ChangeIconDialog brings up the Change Icon OLE dialog box.

```
function ChangeIconDialog: Boolean;
```

Description Call *ChangeIconDialog* to let the user choose a different icon (from .ICO, .DLL, and .EXE files) to use when displaying the OLE object as an icon. An OLE object must already be loaded before calling *ChangeIconDialog*. *ChangeIconDialog* returns *True* if the dialog box was successfully displayed and the user chose the OK button. *ChangeIconDialog* returns *False* otherwise.

See also *Iconic* property, *ObjectPropertiesDialog* method

Close

Close deactivates the OLE object.

```
procedure Close;
```

Description Call *Close* to deactivate the OLE object and terminate its server application, without removing it from the container. Any changes the user made to the OLE object are automatically saved. After calling *Close*, the container's *State* property is *osLoaded*.

Note If *Close* is called when no OLE object is loaded, *TOleContainer* raises an exception.

See also *OnDeactivate* event, *State* property

Copy

Copy copies the OLE object to the Windows clipboard.

```
procedure Copy;
```

Description Call *Copy* to write the OLE object to the clipboard. If there is no OLE object in the container, *Copy* raises an exception.

See also *Paste* method, *PasteSpecialDialog* method, *SaveToFile* method, *SaveToStream* method

Create

Create creates and initializes an instance of *TOleContainer*.

```
constructor Create(AOwner: TComponent); override;
```

Description Call *Create* to instantiate a *TOleContainer* object at runtime. OLE containers placed on forms at design time are created automatically. After calling the inherited constructor, *Create*

- Sets *ControlStyle* to [*csClickEvents, csSetCaption, csOpaque, csDoubleClicks*].
- Sets *TabStop* to *True*.
- Sets *ParentColor* to *False* and *BorderStyle* to *bsSingle*.
- Sets *AllowActiveDoc* and *AllowInPlace* to *True*.
- Sets *AutoActivate* to *aaDoubleClick* and *AutoVerbMenu* to *True*.
- Sets *CopyOnSave* to *True*.

See also *AllowActiveDoc* property, *AllowInPlace* property, *AutoActivate* property, *AutoVerbMenu* property, *BorderStyle* property, *ControlStyle* property, *CopyOnSave* property, *ParentColor* property, *TabStop* property

CreateLinkToFile

CreateLinkToFile creates a linked OLE object from the contents of a file.

```
procedure CreateLinkToFile(const FileName: string; Iconic: Boolean);
```

Description Call *CreateLinkToFile* to create a linked OLE object from the file specified by the *FileName* parameter. *Iconic* specifies whether the object is shown as an icon (*True*) or displayed as it would be in the server application (*False*). If there is already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See also *CreateObject* method, *CreateObjectFromFile* method, *CreateObjectFromInfo* method

CreateObject

CreateObject creates an embedded OLE object using the specified class name. (also known as programmatic identifier).

```
procedure CreateObject(const OleClassName: string; Iconic: Boolean);
```

Description Call *CreateObject* to create an OLE object using the class name (also known as programmatic identifier) specified by the *OleClassName* parameter. The *Iconic* parameter indicates whether the object is shown as an icon (*True*) or displayed as it would be in the server application (*False*). If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See also *CreateLinkToFile* method, *CreateObjectFromFile* method, *CreateObjectFromInfo* method, *OleClassName* property

CreateObjectFromFile

CreateObjectFromFile creates an embedded OLE object from the contents of a file.

```
procedure CreateObjectFromFile(const FileName: string; Iconic: Boolean);
```

Description Call *CreateObjectFromFile* to create an OLE object from the file specified by the *FileName* parameter. The *Iconic* parameter indicates whether the object is shown as an icon (*True*) or displayed as it would be in the server application (*False*). If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See also *CreateLinkToFile* method, *CreateObject* method, *CreateObjectFromInfo* method

CreateObjectFromInfo

CreateObjectFromInfo creates an OLE object matching the specified *TCreateInfo* record.

```
type
  TCreateType = (ctNewObject, ctFromFile, ctLinkToFile, ctFromData, ctLinkFromData);
  TCreateInfo = record
    CreateType: TCreateType;
    ShowAsIcon: Boolean;
    IconMetaPict: HGlobal;
    ClassID: TCLSID;
    FileName: WideString;
    DataObject: IDataObject;
  end;

procedure CreateObjectFromInfo(const CreateInfo: TCreateInfo);
```

Description Call *CreateObjectFromInfo* to create a linked or embedded OLE object based on the specifications in the *CreateInfo* parameter. *CreateInfo* specifies whether the object should be created from a file, from a data transfer object, or from a class identifier, and it provides the necessary supporting information. *CreateInfo* also indicates whether the OLE object should be displayed as an *Icon*, and allows the icon to be specified. If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

Note The *CreateLinkToFile*, *CreateObject*, and *CreateObjectFromFile* methods are oneline calls for the most common types of OLE objects. They can be used instead of filling out the *TCreateInfo* record.

See also *CreateLinkToFile* method, *CreateObject* method, *CreateObjectFromFile* method, *TCreateInfo* type

Destroy

Destroy destroys an instance of *TOleContainer*.

```
destructor Destroy; override;
```

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the OLE container is not already freed, and only then calls *Destroy*. *Destroy* frees the OLE object before calling the inherited destructor.

See also *DestroyObject* method, *Free* method

DestroyObject

DestroyObject destroys the container's OLE object.

```
procedure DestroyObject;
```

Description Call *DestroyObject* to destroy the OLE object, discarding any unsaved changes the user made to it.

See also *Close* method, *OleObject* property

DoVerb

DoVerb requests the OLE object to perform some action.

```
procedure DoVerb(Verb: Integer);
```

Description Call *DoVerb* to request that the OLE object perform one of the actions listed by the *ObjectVerbs* property. OLE defines several verbs, such as *ovShow* (to display the OLE object) and *ovPrimary* (the default action, usually to activate the OLE object). OLE objects can also define their own custom verbs. Use the *ObjectVerbs* property to get a list of those custom verbs.

See also *ObjectVerbs* property

GetIconMetaPict

GetIconMetaPict returns the global memory handle of a metafile containing the OLE object's icon.

```
function GetIconMetaPict: HGlobal;
```

Description Call *GetIconMetaPict* to obtain a metafile handle for drawing the OLE object's icon on another canvas. Use the global *DestroyMetaPict* procedure to destroy

the handle. If the OLE container is empty, *GetIconMetaPict* raises an exception.

See also *DestroyMetaPict* procedure, *TMetafile* object, *TMetafileCanvas* object

InsertObjectDialog

InsertObjectDialog brings up the Insert Object OLE dialog box.

```
function InsertObjectDialog: Boolean;
```

Description Call *InsertObjectDialog* to let the user create an OLE object. The OLE object can be embedded or linked. *InsertObjectDialog* returns *True* if the dialog box was successfully displayed and the user chose the OK button. *InsertObjectDialog* returns *False* otherwise. If the user chose OK and there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See also *CreateObjectFromInfo* method, *ObjectPropertiesDialog* method, *PasteSpecialDialog* method

LoadFromFile

LoadFromFile loads an OLE object from the specified file.

```
procedure LoadFromFile(const FileName: string);
```

Description Call *LoadFromFile* to load an OLE object that was saved using the *SaveToFile* method. If *OldStreamFormat* is *True*, *LoadFromFile* loads files saved by Delphi 1.0's *TOleContainer* object as well as files saved using the current format; if *OldStreamFormat* is *False*, *LoadFromFile* will not load files saved by Delphi 1.0. If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See also *LoadFromStream* method, *OldStreamFormat* property, *SaveToFile* method

LoadFromStream

LoadFromStream loads an OLE object from the specified stream.

```
procedure LoadFromStream(Stream: TStream);
```

Description Call *LoadFromStream* to load an OLE object from a stream. If *OldStreamFormat* is *True*, *LoadFromStream* loads OLE objects saved by Delphi 1.0's *TOleContainer* object as well as OLE objects saved using the current format; if *OldStreamFormat* is *False*, *LoadFromStream* will not load OLE objects saved by Delphi 1.0. If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See also *LoadFromFile* method, *OldStreamFormat* property, *SaveToStream* method, *TStream* object

ObjectPropertiesDialog

ObjectPropertiesDialog brings up the OLE Object Properties property sheet.

```
function ObjectPropertiesDialog: Boolean;
```

Description Call *ObjectPropertiesDialog* to let the user see and edit the properties of the OLE object. Any changes the user makes in the property sheet are automatically made to the OLE object. *ObjectPropertiesDialog* returns *True* if the property sheet was successfully displayed and the user chose the OK button. It returns *False* otherwise.

If the OLE container is empty, *ObjectPropertiesDialog* raises an exception.

See also *DoVerb* method

Paste

Paste pastes the contents of the Windows clipboard as an embedded object.

```
procedure Paste;
```

Description Call *Paste* to embed the object on the clipboard in the OLE container. If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See also *CanPaste* property, *CreateObjectFromInfo* method, *PasteSpecialDialog* method

PasteSpecialDialog

PasteSpecialDialog brings up the Paste Special OLE dialog box.

```
function PasteSpecialDialog: Boolean;
```

Description Call *PasteSpecialDialog* to let the user control how the contents of the Windows clipboard are pasted into the container. The Paste Special dialog box lets the user select the format of the data, whether it should be embedded or linked, and whether the OLE object should be displayed as an icon (and if so, to choose a different icon). *PasteSpecialDialog* returns *True* if the dialog box was successfully displayed and the user chose the OK button. It returns *False* otherwise. If the user chose the OK button when there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See also *CanPaste* property, *InsertObjectDialog* method, *Paste* method

Run

Run runs the OLE object's server application, but doesn't open or activate the OLE object itself.

```
procedure Run;
```

Description Call *Run* to ensure that the server application is running. Once the server application is running, opening and activating the OLE object happens much more quickly. An OLE object must already be loaded in the container before calling *Run*, Otherwise, *Run* raises an exception.

See also *State* property

SaveToFile

SaveToFile saves the OLE object to the specified file.

```
procedure SaveToFile(const FileName: string);
```

Description Call *SaveToFile* to save the OLE object. If *CopyOnSave* is *True*, *SaveToFile* first creates a temporary OLE object that compresses redundant data, saving space in the file that is saved. If *OldStreamFormat* is *True*, *SaveToFile* writes a file that can be read by Delphi 1.0's *TOLEContainer* component.

An OLE object must already be loaded in the container before calling *SaveToFile*. Otherwise, *SaveToFile* raises an exception.

See also *Copy* method, *CopyOnSave* property, *LoadFromFile* method, *OldStreamFormat* property, *SaveToStream* method

SaveToStream

SaveToStream saves the OLE object to the specified stream.

```
procedure SaveToStream(Stream: TStream);
```

Description Call *SaveToStream* to save the OLE object. If *CopyOnSave* is *True*, *SaveToStream* first creates a temporary OLE object that compresses redundant data, reducing the information that must be saved. If *OldStreamFormat* is *True*, *SaveToStream* writes the object in a format that can be read by Delphi 1.0's *TOLEContainer* component.

An OLE object must already be loaded in the container before calling *SaveToStream*. Otherwise, *SaveToStream* raises an exception.

See also *Copy* method, *CopyOnSave* property, *LoadFromStream* method, *OldStreamFormat* property, *SaveToFile* method

UpdateObject

UpdateObject refreshes the OLE object.

```
procedure UpdateObject;
```

Description Linked OLE objects and embedded objects that contain linked OLE objects get out of date when the source of the link is updated. *UpdateObject* rereads the source to ensure that the OLE object has current data. If there is no OLE object loaded in the container, *UpdateObject* has no effect.

See also *Linked* property, *UpdateVerbs* method

UpdateVerbs

UpdateVerbs refreshes the list of verbs the OLE object can respond to.

```
procedure UpdateVerbs;
```

Description *TOleContainer* automatically calls *UpdateVerbs* when the *ObjectVerbs* property is first read, but some OLE objects change their verbs when they perform some actions. The Media Clip OLE object, for example, changes its *Play* verb to *Stop* while running. Call *UpdateVerbs* to ensure that the list of verbs is appropriate to the state of the OLE object.

An OLE object must already be loaded in the container before calling *UpdateVerbs*. Otherwise, *UpdateVerbs* raises an exception.

See also *AutoVerbMenu* property, *ObjectVerbs* property

TOleControl

Unit: OleCtrls

TObject • TPersistent • TComponent • TControl • TWinControl • TOleControl

TOleControl is the base class from which OCX controls are derived.

Description *TOleControl* is derived from *TWinControl* and handles the interactions with OLE necessary for using OCX controls in the Delphi environment. It is unlikely that you will ever need to derive a component from this class.

See also *TOleContainer* object

Properties

The following table lists the properties of *TOleControl*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent

Property	Visibility	Default	Documented in
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
Enabled	public	True	TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
Name	published		TOleControl
OleObject	public, read-only		TOleControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentWindow	public		TWinControl
ShowHint	public		TControl
Showing	public, read-only		TWinControl
TabOrder	public	–1	TWinControl
TabStop	public	True	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	public	True	TControl
Width	published		TControl
WindowProc	public		TControl

Name

Name is the name of the associated OCX control.

property Name: TComponentName;

Description *Name* is used to get the name of the OCX control associated with the OLE control object.

OleObject

OleObject is the OCX control associated with the *TOleControl* object.

property OleObject: Variant;

Description *OleObject* is used to get the OCX that is associated with the *TOleControl*.

Methods

The following table lists the methods of *TOleControl*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
BrowseProperties	public	TOleControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TOleControl
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TOleControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DoObjectVerb	public	TOleControl
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetEnumPropDesc	public	TOleControl
GetHelpContext	public	TOleControl
GetInterface	public	TObject

Method	Visibility	Documented in
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetObjectVerbs	public	TOleControl
GetParentComponent	public	TComponent
GetPropDisplayString	public	TOleControl
GetPropDisplayStrings	public	TOleControl
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
IsCustomProperty	public	TOleControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TOleControl
SetFocus	public	TWinControl
SetPropDisplayString	public	TOleControl
SetTextBuf	public	TControl
Show	public	TControl
ShowAboutBox	public	TOleControl

Method	Visibility	Documented in
Update	public	TWinControl
UpdateControlState	public	TWinControl

BrowseProperties

BrowseProperties notifies the OCX control to display its Properties dialog box.

```
procedure BrowseProperties;
```

Description Use *BrowseProperties* to notify the OCX control to display its Properties dialog box. The Properties dialog box contains property pages for the OCX control.

Create

Create creates a new instance of the *TOleControl* object.

```
constructor Create(AOwner: TComponent); override;
```

Description Use *Create* to create a new instance of the *TOleControl* object. *Create* calls the inherited *Create* method and creates and initializes internal objects.

See also *TWinControl.Create* method

Destroy

Destroy frees the memory associated with this instance of the *TOleControl* object.

```
destructor Destroy; override;
```

Description Do not call *Destroy*. Instead, call *Free*, which calls *Destroy* if this instance of the *TOleControl* object is not **nil**.

See also *Free* method

DoObjectVerb

DoObjectVerb notifies the OCX control to perform one of its actions.

```
procedure DoObjectVerb(Verb: Integer);
```

Description Use *DoObjectVerb* to notify the OCX control to perform one of its actions. *Verb* is an integer that corresponds to the action.

For more information about verbs, search the OLE.HLP file or the latest Microsoft Developer Network CD.

GetEnumPropDesc

GetEnumPropDesc returns descriptions of the OCX control's properties.

```
function GetEnumPropDesc(DispID: Integer): TEnumPropDesc;
```

Description Use *GetEnumPropDesc* to return descriptions of the OCX control's properties.

GetEnumPropDesc returns the descriptions in a *TEnumPropDesc* object. For information about the *TEnumPropDesc* object, refer to the OLECTRLS.PAS file.

GetHelpContext

GetHelpContext retrieves the Help context of a member of the OCX control.

```
function GetHelpContext(Member: string; var HelpCtx: Integer; var HelpFile: string): Boolean;
```

Description Use *GetHelpContext* to retrieve the Help context of a member of the OCX control.

Member is the name of a property, method, or event of the OCX control. To retrieve the help context of the OCX control itself, set *Member* to **nil**.

HelpCtx is a variable in which the Help context identifier of *Member* is to be stored.

HelpFile is a variable in which the name of the Help file containing *HelpCtx* is to be stored.

The information about the OCX control is retrieved from its *ITypeInfo* interfaces. For more information, search the OLE.HLP file for ITypeInfo.

GetHelpContext returns *False* if *Member* is not found; otherwise, *GetHelpContext* returns *True*.

GetObjectVerbs

GetObjectVerbs retrieves the names of the verbs implemented by the OCX control.

```
procedure GetObjectVerbs(List: TStrings);
```

Description Use *GetObjectVerbs* to retrieve the names of the verbs implemented by the OCX control.

List is where the names of the verbs are to be stored.

GetPropDisplayString

GetPropDisplayString returns the display string for an interface for the OCX control.

```
function GetPropDisplayString(DispID: Integer): string;
```

Description *GetPropDisplayString* returns the display string for an interface for the OCX control.

DispID is the identifier for the interface. For more information, search the OLE.HLP file for DISPID.

For more information about *GetPropDisplayString*, search the latest Microsoft Developer Network CD for *GetDisplayStrings*.

See also *GetPropDisplayStrings* method, *SetPropDisplayString* method

GetPropDisplayStrings

GetPropDisplayStrings returns the display strings for an interface for the OCX control.

```
procedure GetPropDisplayStrings(DispID: Integer; List: TStrings);
```

Description *GetPropDisplayString* returns the display strings for an interface for the OCX control.

DispID is the identifier for the interface. For more information, search the OLE.HLP file for DISPID.

List is where the strings are to be stored.

For more information about *GetPropDisplayStrings*, search the latest Microsoft Developer Network CD for GetDisplayStrings.

See also *GetPropDisplayString* method

IsCustomProperty

IsCustomProperty indicates whether an interface for the OCX control is a custom property.

```
function IsCustomProperty(DispID: Integer): Boolean;
```

Description Use *IsCustomProperty* to determine whether an interface for the OCX control is a custom property.

DispID is the identifier for the interface. For more information, search the OLE.HLP file for DISPID.

IsCustomProperty returns *True* if an interface is a custom property; otherwise, *IsCustomProperty* returns *False*.

SetBounds

SetBounds sets the boundaries of the OCX control.

```
procedure SetBounds(ALeft, ATop, AWidth, AHeight: Integer); override;
```

Description Use *SetBounds* to set the boundaries of the OCX control.

ALeft, *ATop*, *AWidth*, and *AHeight* are the x-coordinate of the upper-left corner, the y-coordinate of the upper-left corner, and the width and the height of the OCX control.

SetPropDisplayString

SetPropDisplayString sets the display string for an interface for the OCX control.

```
procedure SetPropDisplayString(DispID: Integer; const Value: string);
```

Description Use *SetPropDisplayString* to set the display string for an interface for the OCX control.

DispID is the identifier for the interface. For more information, search the OLE.HLP file for DISPID.

Value is the new display string for the interface.

See also *GetPropDisplayString* method

ShowAboutBox

ShowAboutBox notifies the OCX control to display its About box.

```
procedure ShowAboutBox;
```

Description Use *ShowAboutBox* to notify the OCX control to display its About box.

TOpenDialog

Unit: Dialogs Palette: Dialogs

TObject • TPersistent • TComponent • TCommonDialog • TOpenDialog

TOpenDialog displays a file-selection dialog.

Description *TOpenDialog* displays a modal Windows dialog box for selecting and opening files. The dialog does not appear at runtime until it is activated by a call to the *Execute* method. When the user clicks Open, the dialog closes and the selected file or files are stored in the *Files* property.

See also *TOpenPictureDialog* object, *TSaveDialog* object, *TSavePictureDialog* object

Properties

The following table lists the properties of *TOpenDialog*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent

Property	Visibility	Default	Documented in
ComponentStyle	public, read-only		TComponent
Ctl3D	published	True	TCommonDialog
DefaultExt	published		TOpenDialog
DesignInfo	public		TComponent
FileEditStyle	published		TOpenDialog
FileName	published		TOpenDialog
Files	public, read-only		TOpenDialog
Filter	published		TOpenDialog
FilterIndex	published	1	TOpenDialog
HelpContext	published	0	TCommonDialog
HistoryList	public		TOpenDialog
InitialDir	published		TOpenDialog
Name	published		TComponent
Options	published	[]	TOpenDialog
Owner	public, read-only		TComponent
Tag	published	0	TComponent
Title	published		TOpenDialog
VCLComObject	public		TComponent

DefaultExt

Specifies a default file extension.

```
property DefaultExt: string;
```

Description *DefaultExt* specifies a file extension that is appended automatically to the selected file name, unless the selected file name already includes a registered extension. If the user selects a file name with an extension that is unregistered, *DefaultExt* is appended to the unregistered extension.

Extensions longer than three characters are not supported. Do not include the period (.) that divides the file name and its extension.

FileEditStyle

Determines the style of the file-selection dialog. (Obsolete.)

```
type TFileEditStyle = (fsEdit, fsComboBox);
property FileEditStyle: TFileEditStyle;
```

Description *FileEditStyle* is maintained for compatibility with older versions of Delphi.

FileName

Returns the name and directory path of the last file selected.

type TFileName = **string**;

property FileName: TFileName;

Description The *FileName* property returns the name and complete directory path of the most recently selected file. The value of *FileName* is the same as the first item in the *Files* property.

To make a file name appear by default in the dialog's edit box, assign a value to *FileName* in the Object Inspector or in program code. Programmatic changes to *FileName* have no effect while the dialog is active.

See also *Files* property

Files

Returns a list of the selected file names.

property Files: TStrings;

Description *FileName* is a *TStrings* instance that contains each selected file name with its full directory path. (To let users select multiple file names, set the *ofAllowMultiSelect* flag in *Options*.)

See also *FileName* property, *Options* property, *TStrings* object

Filter

Determines the file masks (filters) available in the dialog.

property Filter: **string**;

Description The file-selection dialog includes a drop-down list of file types under the edit box. When the user picks a file type from the list, only files of the selected type are displayed in the dialog.

To configure file masks at design time, click on the ellipsis marks (...) to the right of the *Filter* property in the Object Inspector. This opens the Filter editor. In the left column of the Filter editor, under Filter Name, type a brief description of each file type that will be available at runtime. In the right column, under Filter, type the file mask corresponding to each description. For example, the description "Text files" might appear to the left of the mask "*.TXT", and the description "Pascal source files" might appear to the left of the mask "*.PAS". Since the description appears in the drop-down list at runtime, it is often helpful to show the mask explicitly in the description (for example, "Text files (*.txt)").

To create file masks in program code, assign a value to the *Filter* property that consists of a description and a mask separated by a vertical bar (pipe) character. Do not include spaces around the vertical bar. For example,

```
OpenDialog1.Filter := 'Text files (*.txt)|*.TXT'
```

Multiple filters should be separated by vertical bars. For example,

```
OpenDialog1.Filter := 'Text files (*.txt)|*.TXT|Pascal files (*.pas)|*.PAS'
```

To include multiple masks in a single filter, separate the masks with semicolons. This works both in the Object Inspector and in program code. For example,

```
OpenDialog1.Filter := 'Pascal files|*.PAS;*.DPK;*.DPR'
```

If no value is assigned to *Filter*, the dialog displays all file types.

See also *FilterIndex* property, *GraphicFilter* function

FilterIndex

Determines which filter is selected by default when the dialog opens.

property FilterIndex: Integer;

Description *FilterIndex* determines which of the file types in *Filter* is selected by default when the dialog opens. Set *FilterIndex* to 1 to choose the first file type in the list as the default, or set *FilterIndex* to 2 to choose the second file type as the default, and so forth. If the value of *FilterIndex* is out or range, the first file type listed in *Filter* is the default.

See also *Filter* property

HistoryList

Maintains a list of previously selected files. (Obsolete.)

property HistoryList: TStrings;

Description *HistoryList* is maintained for compatibility with older versions of Delphi.

See also *TStrings* object

InitialDir

Determines the current directory when the dialog opens.

property InitialDir: **string**;

Description *InitialDir* determines the default directory displayed in the file-selection dialog when it opens. For example, to point the dialog at the WINDOWS\SYSTEM directory, set the value of *InitialDir* to C:\WINDOWS\SYSTEM.

If no value is assigned to *InitialDir*, or if the specified directory does not exist, the dialog opens with the current working directory displayed.

Options

Determines the appearance and behavior of the file-selection dialog.

```
type
  TOpenOption = (ofReadOnly, ofOverwritePrompt, ofHideReadOnly, ofNoChangeDir,
    ofShowHelp, ofNoValidate, ofAllowMultiSelect, ofExtensionDifferent, ofPathMustExist,
    ofFileMustExist, ofCreatePrompt, ofShareAware, ofNoReadOnlyReturn, ofNoTestFileCreate,
    ofNoNetworkButton, ofNoLongNames, ofOldStyleDialog, ofNoDereferenceLinks);
  TOpenOptions = set of TOpenOption;

property Options: TOpenOptions;
```

Description Use the *Options* property to customize the appearance and functionality of the dialog. The possible values of *Options* are

Value	Meaning
ofAllowMultiSelect	Allows users to select more than one file in the dialog.
ofCreatePrompt	Generates a warning message if the user tries to select a nonexistent file, asking whether to create a new file with the specified name.
ofExtensionDifferent	This flag is turned on at runtime whenever the selected filename has an extension that differs from *DefaultExt*. If you use this flag an application, remember to reset it.
ofFileMustExist	Generates an error message if the user tries to select a nonexistent file.
ofHideReadOnly	Removes the Open As Read Only check box from the dialog.
ofNoChangeDir	After the user clicks OK, resets the current directory to whatever it was before the file-selection dialog opened.
ofNoDereferenceLinks	Disables dereferencing of Windows shortcuts. If the user selects a shortcut, assigns to *FileName* the path and file name of the shortcut itself (the .LNK file), rather than the file linked to the shortcut.
ofNoLongNames	Displays 8.3-character file names only.
ofNoNetworkButton	Removes the Network button (which opens a Map Network Drive dialog) from the file-selection dialog. Applies only if the *ofOldStyleDialog* flag is on.
ofNoReadOnlyReturn	Generates an error message if the user tries to select a read-only file.
ofNoTestFileCreate	Disables checking for network file protection and inaccessibility of disk drives. Applies only when the user tries to save a file in a create-no-modify shared network directory.
ofNoValidate	Disables checking for invalid characters in file names. Allows selection of file names with invalid characters.
ofOldStyleDialog	Creates the older style of file-selection dialog.
ofOverwritePrompt	Generates a warning message if the user tries to select a file name that is already in use, asking whether to overwrite the existing file. (Use with *TSaveDialog* and *TSavePictureDialog*.)
ofPathMustExist	Generates an error message if the user tries to select a file name with a nonexistent directory path.

Value	Meaning
ofReadOnly	Selects the Open As Read Only check box by default when the dialog opens.
ofShareAware	Ignores sharing errors and allows files to be selected even when sharing violations occur.
ofShowHelp	Displays a Help button in the dialog.

By default, all options flags are off.

See also *DefaultExt* property, *FileName* property, *TSaveDialog* object, *TSavePictureDialog* object

Title

Specifies the text in the dialog's title bar.

```
property Title: string;
```

Description Use *Title* to specify the text that appears in the file-selection dialog's title bar. If no value is assigned to *Title*, the dialog has the title "Open".

Events

The following table lists the events of *TOpenDialog*:

Event	Visibility	Documented in
OnClose	published	TCommonDialog
OnFolderChange	published	TOpenDialog
OnSelectionChange	published	TOpenDialog
OnShow	published	TCommonDialog
OnTypeChange	published	TOpenDialog

OnFolderChange

OnFolderChange occurs when a directory is opened or closed from the dialog.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnFolderChange: TNotifyEvent;
```

Description The *OnFolderChange* event occurs when the user changes the directory whose contents are displayed in the dialog. This can happen when the user double-clicks on a directory, clicks the Up arrow, or uses the list box at the top of the dialog to navigate through the directory structure.

OnSelectionChange

OnSelectionChange occurs when file names displayed in the dialog are changed.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnSelectionChange: TNotifyEvent;
```

Description The *OnSelectionChange* event occurs when the user does something to change the list displayed in the dialog. This can include opening the file-selection dialog, highlighting a file or directory, selecting a new filter, selecting a new directory, or creating a new folder.

See also *FileName* property

OnTypeChange

OnTypeChange occurs when the file types displayed in the dialog are changed.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnTypeChange: TNotifyEvent;
```

Description The *OnTypeChange* event occurs when the user selects a new filter from the Files of Type list box at the bottom of the dialog.

See also *Filter* property

O

Methods

The following table lists the methods of *TOpenDialog*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TOpenDialog
DefaultHandler	public	TObject
Destroy	public	TOpenDialog
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
Execute	public	TOpenDialog

Method	Visibility	Documented in
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetStaticRect	protected	TOpenDialog
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Create

Creates and initializes a *TOpenDialog* instance.

constructor Create(AOwner: TComponent); **override**;

Description The *Create* method generates a *TOpenDialog* instance, but the new dialog does not appear on the form at runtime until the *Execute* method is called.

See also *Execute* method

Destroy

Destroys the *TOpenDialog* and frees its memory.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Call *Free* instead. *Free* verifies that the file-selection dialog is not already destroyed before calling *Destroy*.

See also *Free* method

Execute

Displays the file-selection dialog.

function Execute: Boolean; **override**;

Description *Execute* opens the file-selection dialog, returning *True* when the user selects a file and clicks Open. If the user clicks Cancel, *Execute* returns *False*.

GetStaticRect

Returns the coordinates of the reserved area of the dialog box.

function GetStaticRect: TRect; **virtual**;

Description *GetStaticRect* is provided for components that descend from *TOpenDialog* and require the placement of new controls alongside the standard ones inherited from the parent class. For Explorer-style dialogs (running under Windows 95 or NT 4.0), *GetStaticRect* returns the size and location of the standard controls within the dialog. For older versions of Windows, or if *ofOldStyleDialog* is enabled in *Options*, *GetStaticRect* returns the client dimensions of the entire dialog box.

The following example is taken from *TOpenPictureDialog*. It calculates the preview rectangle based on the difference between the client area of the dialog and the static area of the standard controls.

```
procedure TOpenPictureDialog.DoShow;
var
  PreviewRect, StaticRect: TRect;
begin
  { Set preview area to entire dialog }
  GetClientRect(Handle, PreviewRect);
  StaticRect := GetStaticRect;
  { Move preview area to right of static area }
  PreviewRect.Left := StaticRect.Left + (StaticRect.Right - StaticRect.Left);
...
```

See also *Options* property, *TCommonDialog.Template* property, *TOpenPictureDialog* object, *TRect* type

TOpenPictureDialog

Unit: ExtDlgs | Palette: Dialogs

TObject • TPersistent • TComponent • TCommonDialog • TOpenDialog • TOpenPictureDialog

TOpenPictureDialog displays a graphics-file selection dialog.

Description *TOpenPictureDialog* displays a modal Windows dialog box for selecting and opening graphics files. This component is just like *TOpenDialog*, except that it includes a rectangular preview region. If the selected image can be read by

TPicture, it is displayed in the preview region; supported file types include bitmap (.BMP), icon (.ICO), Windows metafile (.WMF), and enhanced Windows metafile (.EMF). If the selected image cannot be displayed, "(None)" appears in the preview region.

Note *TPicture* raises an *EInvalidGraphic* exception when the user selects a file of an unrecognized type. Expect errors of this kind when debugging with Break on Exception enabled.

See also *EInvalidGraphic* object, *TOpenDialog* object, *TPicture* object, *TSavePictureDialog* object

Properties

The following table lists the properties of *TOpenPictureDialog*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Ctl3D	published	True	TCommonDialog
DefaultExt	published		TOpenDialog
DesignInfo	public		TComponent
FileEditStyle	published		TOpenDialog
FileName	published		TOpenDialog
Files	public, read-only		TOpenDialog
Filter	published		TOpenDialog
FilterIndex	published	1	TOpenDialog
HelpContext	published	0	TCommonDialog
HistoryList	public		TOpenDialog
InitialDir	published		TOpenDialog
Name	published		TComponent
Options	published	[]	TOpenDialog
Owner	public, read-only		TComponent
Tag	published	0	TComponent
Title	published		TOpenDialog
VCLComObject	public		TComponent

Events

The following table lists the events of *TOpenPictureDialog*:

Event	Visibility	Documented in
OnClose	published	TCommonDialog
OnSelectionChange	published	TOpenDialog
OnShow	published	TCommonDialog

Methods

The following table lists the methods of *TOpenPictureDialog*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TOpenPictureDialog
DefaultHandler	public	TObject
Destroy	public	TOpenPictureDialog
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
Execute	public	TOpenPictureDialog
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent

Method	Visibility	Documented in
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Create

Creates and initializes a *TOpenPictureDialog* instance.

constructor Create(AOwner: TComponent); **override**;

Description The *Create* method generates a *TOpenPictureDialog* instance, but the dialog does not appear at runtime until the *Execute* method is called. *Create* initializes the *Filter* property by defining a filter for each file type that *TPicture* supports.

See also *Execute* method, *Filter* property, *TPicture* object

Destroy

Destroys the *TOpenPictureDialog* and frees its memory.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Call *Free* instead. *Free* verifies that the file-selection dialog is not already destroyed before calling *Destroy*.

See also *Free* method

Execute

Displays the file-selection dialog.

function Execute: Boolean; **override**;

Description *Execute* opens the file-selection dialog, returning *True* when the user selects a file and clicks Open. If the user clicks Cancel, *Execute* returns *False*.

TPageControl

Unit: ComCtrls Palette: Win32

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomTabControl • TPageControl

TPageControl is a set of pages used to make a multiple page dialog box.

Description Use *TPageControl* to create a multiple page dialog or tabbed notebook. *TPageControl* displays multiple overlapping pages which are *TTabSheet* objects. The user selects a page by clicking the page's tab that appears at the top of the control. To add a new page to a *TPageControl* object at design time, right-click the *TPageControl* object and choose New Page.

To create a tabbed control that uses only a single body portion (page), use *TTabControl* instead.

See also *TTabControl* object, *TTabSheet* object

Properties

The following table lists the properties of *TPageControl*:

Property	Visibility	Default	Documented in
ActivePage	published		TPageControl
Align	published	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
HotTrack	published	False	TCustomTabControl

Property	Visibility	Default	Documented in
Left	published		TControl
MultiLine	published	False	TCustomTabControl
Name	published		TControl
Owner	public, read-only		TComponent
PageCount	public, read-only		TPageControl
Pages	public, read-only		TPageControl
Parent	public		TControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ScrollOpposite	published	False	TCustomTabControl
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabHeight	published	0	TCustomTabControl
TabOrder	published	–1	TWinControl
TabPosition	published	tpTop	TCustomTabControl
TabStop	published	True	TWinControl
TabWidth	published	0	TCustomTabControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

ActivePage

ActivePage is the page currently displayed by the page control.

property ActivePage: TTabSheet;

Description Read *ActivePage* to determine which page is selected by the user. The user selects a page by clicking the mouse on its tab. Set *ActivePage* to bring a page to the foreground. Only one page can be active at a time.

To change the *ActivePage* to the page that precedes or follows the active page, use the *SelectNextPage* method. To iterate through all the pages to locate a specific page, use the *FindNextPage* method.

See also *FindNextPage* method, *SelectNextPage* method, *TTabSheet* object

PageCount

PageCount is the number of pages in the *TPageControl* object.

```
property PageCount: Integer;
```

Description Use *PageCount* to determine the number of pages listed by the *Pages* property. If the *MultiLine* property is *False*, not all pages may have tabs visible to the user.

See also *MultiLine* property, *Pages* property

Pages

Pages is an indexed array of all the pages in the *TPageControl*.

```
property Pages[Index: Integer]: TTabSheet;
```

Description Use *Pages* to gain direct access to a particular page in the page control. For example, use *Pages* to disable individual pages in the page control. Specify which page to access using *Index*, where an *Index* of 0 specifies the first page, an *Index* of 1 specifies the second page, and so on. Each page is a *TTabSheet* object.

To locate a page in the page control by its position relative to another page, use the *FindNextPage* method.

See also *ActivePage* property, *FindNextPage* method, *PageCount* property, *TTabSheet* object

Events

The following table lists the events of *TPageControl*:

Event	Visibility	Documented in
OnChange	published	TCustomTabControl
OnChanging	published	TCustomTabControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TPageControl*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TPageControl
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TPageControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
FindNextPage	public	TPageControl
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent

Method	Visibility	Documented in
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SelectNextPage	public	TPageControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

P

Create

Create creates an instance of *PageControl*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to instantiate a page control at runtime. Page controls added to forms at design time are created automatically. After calling the inherited constructor, *Create* sets *ControlStyle* to *csDoubleClicks*, and creates a *TList* object to hold the pages of the control.

When creating a page control at runtime, the pages must be added programmatically. To add a page to a page control at runtime, create the *TTabSheet* object for the page and set its *PageControl* property.

See also *ControlStyle* property, *TCustomTabControl.Create* method, *TTabSheet* object

Destroy

Destroy destroys an instance of *TPageControl*.

```
destructor Destroy; override;
```

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the page control is not already freed, and only then calls *Destroy*. *Destroy* frees the *TList* object that holds the pages of the control, and then calls the inherited destructor.

See also *Free* method

FindNextPage

FindNextPage returns the next page in the page control before or after a specified page.

```
function FindNextPage(CurPage: TTabSheet; GoForward, CheckTabVisible: Boolean):
  TTabSheet;
```

Description Call *FindNextPage* to locate a page in the page control by its position relative to the page specified by the *CurPage* parameter. If the *GoForward* parameter is *True*, *FindNextPage* returns the next page in the page control after *CurPage*. If *GoForward* is *False*, *FindNextPage* returns the previous page. If *CurPage* is not a page in the page control, *FindNextPage* returns the first page when *GoForward* is *True* or the last page when *GoForward* is *False*. If *GoForward* is *True* and *CurPage* is the last page in the page control, *FindNextPage* returns the first page. If *GoForward* is *False* and *CurPage* is the first page in the page control, *FindNextPage* returns the last page.

When *CheckTabVisible* is *True*, *FindNextPage* returns the next page with a *TabVisible* property set to *True*; if *CheckTabVisible* is *False*, *FindNextPage* returns the next page, including pages with *TabVisible* set to *False*.

See also *Pages* property, *SelectNextPage* method, *TTabSheet* object, *TTabSheet.TabVisible* property

SelectNextPage

SelectNextPage changes the *ActivePage* to the first visible page that is before or after the currently active page.

```
procedure SelectNextPage(GoForward: Boolean);
```

Description Use *SelectNextPage* to select the page next to the currently active page. If the *GoForward* parameter is *True*, *SelectNextPage* selects the first page after *ActivePage*; if *GoForward* is *False*, *SelectNextPage* selects the first page previous to *ActivePage*. *SelectNextPage* will only considers pages with their *TabVisible* property set to *True*.

Note To select a page at design-time, right-click the *TPageControl* and choose Previous Page or Next Page.

See also *ActivePage* property, *FindNextPage* method, *TTabSheet.TabVisible* property

TPageProducer

Unit: HTTPApp | Palette: Internet

TObject • TPersistent • TComponent • TCustomContentProducer • TCustomPageProducer • TPageProducer

TPageProducer produces a string of HTML commands based on an input template.

Description Use *TPageProducer* to convert an HTML template into a string of HTML commands that can be interpreted by a client application such as a Web browser. HTML templates include HTML commands and HTML-transparent tags that are replaced with customized content by the *OnHTMLTag* event.

See also *TDataSetTableProducer* object, *TQueryTableProducer* object

Properties

The following table lists the properties of *TPageProducer*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DesignInfo	public		TComponent
Dispatcher	published		TCustomContentProducer

Property	Visibility	Default	Documented in
HTMLDoc	published		TCustomPageProducer
HTMLFile	published		TCustomPageProducer
Name	published		TComponent
Owner	public, read-only		TComponent
Tag	published	0	TComponent
VCLComObject	public		TComponent

Events

The following table lists the events of *TPageProducer*:ss

Event	Visibility	Documented in
OnHTMLTag	published	TPageProducer

OnHTMLTag

OnHTMLTag occurs when the page producer needs to render the value of an HTML-transparent tag in the HTML template.

```
type
  TTag = (tgCustom, tgLink, tgImage, tgTable, tgImageMap, tgObject, tgEmbed);
  THTMLTagEvent = procedure (Sender: TObject; Tag: TTag; const TagString: string;
    TagParams: TStrings; var ReplaceText: string) of object;

property OnHTMLTag: THTMLTagEvent;
```

Description Write an *OnHTMLTag* event handler to convert an HTML-transparent tag into the corresponding HTML commands that can be interpreted by a Web browser. If an *OnHTMLTag* event is not supplied, the HTML-transparent tags in the HTML template will be replaced by empty strings.

The *Tag* parameter indicates the type of HTML-transparent tag that was encountered in the HTML template. It can have one of the following values:

Value	Meaning
tgLink	The *TagString* parameter is LINK. The *TagParams* describe a hypertext link. The event handler should return a sequence of HTML commands beginning with an <A> tag and ending with an </A> tag.
tgImage	The *TagString* parameter is IMAGE. The *TagParams* describe a graphic image. The event handler should return an HTML <IMG> tag.
tgTable	The *TagString* parameter is TABLE. The *TagParams* describe a desired tabular image. The event handler should return a sequence of HTML commands beginning with a <TABLE> tag and ending with a </TABLE> tag.
tgImageMap	The *TagString* parameter is IMAGEMAP. The *TagParams* describe a graphic image with associated hotzones. The event handler should return a sequence of HTML commands beginning with a <MAP> tag and ending with a </MAP> tag.

Value	Meaning
tgObject	The *TagString* parameter is OBJECT. The *TagParams* contain a reference to an *ActiveX* control. The event handler should return a sequence of HTML commands beginning with an <OBJECT> tag and ending with an </OBJECT> tag.
tgEmbed	The TagString parameter is EMBED. The *TagParams* contain a reference to a Netscape-compliant add-in DLL. The event handler should return a sequence of HTML commands beginning with an <EMBED> tag and ending with an </EMBED> tag.
tgCustom	The TagString parameter is an application-defined string. The event handler can convert the HTML-transparent tag in an application-specific manner.

The *TagString* parameter is the tag name of the HTML-transparent tag. If the *Tag* parameter is *tgCustom*, *TagString* should be used to determine what conversion to perform. The *TagParams* parameter gives the parameter portion of the HTML-transparent tag. Each string in the *TStrings* object has the form ParamName=Value. Use the *Names* and *Values* properties of the *TStrings* object to parse these parameters.

The *OnHTMLTag* event handler should return the converted HTML commands in the *ReplaceText* parameter. The returned value can contain HTML-transparent tags for another page producer to convert. For example, one page producer might have a template that represents the format of the final HTTP response. It could read the HTTP request and for each section in the request assemble a set of HTML-transparent tags with parameters based on the request. Another page producer might take the content of the first page producer, and interpret those parameters.

See also *DoTagEvent* method, *HandleTag* method, *TStrings.Names* property, *TStrings.Values* property

P

Methods

The following table lists the methods of *TPageProducer*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Content	public	TCustomPageProducer
ContentFromStream	public	TCustomPageProducer
ContentFromString	public	TCustomPageProducer
Create	public	TCustomPageProducer

Method	Visibility	Documented in
DefaultHandler	public	TObject
Destroy	public	TCustomPageProducer
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

TPaintBox

Unit: ExtCtrls | Palette: System

TObject • TPersistent • TComponent • TControl • TGraphicControl • TPaintBox

TPaintBox is a simple graphic control that provides a canvas for applications to use for rendering an image.

Description Use *TPaintBox* to add custom images to a form. Unlike *TImage*, which displays an image that is stored in a bitmap, icon, or metafile, *TPaintBox* requires an application to draw the image directly on a canvas. Use the *OnPaint* event handler to draw on the paint box's *Canvas*, the drawing surface of the paint box.

See also *TCanvas* object, *TImage* object

Properties

The following table lists the properties of *TPaintbox*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
BoundsRect	public		TControl
Canvas	public, read-only		TGraphicControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TControl
ClientRect	public, read-only		TControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Font	published		TControl
Height	published		TControl
Hint	published		TControl
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	True	TControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
PopupMenu	published		TControl
ShowHint	published		TControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl

Property	Visibility	Default	Documented in
Width	published		TControl
WindowProc	public		TControl

Events

The following table lists the events of *TPaintbox*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnPaint	published	TPaintBox
OnStartDrag	published	TControl

OnPaint

OnPaint occurs when the paintbox receives a Windows paint message.

```
property OnPaint: TNotifyEvent;
```

Description Write an *OnPaint* event handler to draw the image of the paint box. Use the methods of the canvas to construct the image that the paintbox presents on a form. If an *OnPaint* event handler is not provided, the paintbox can't be seen on the form at runtime.

See also *Canvas* property

Methods

The following table lists the methods of *TPaintbox*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject

Method	Visibility	Documented in
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
Create	public	TPaintBox
DefaultHandler	public	TObject
Destroy	public	TGraphicControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTextBuf	public	TControl
GetTextLen	public	TControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
Invalidate	public	TControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Paint	protected	TPaintBox
Perform	public	TControl
Refresh	public	TControl
RemoveComponent	public	TComponent
Repaint	public	TControl
SafeCallException	public	TComponent

Method	Visibility	Documented in
ScreenToClient	public	TControl
SendToBack	public	TControl
SetBounds	public	TControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TControl

Create

Create creates an instance of *TPaintBox*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to instantiate a *TPaintBox* object at runtime. *TPaintBox* objects added to forms at design time are created automatically.

Create allocates memory and sets the default values for the instance of the *TPaintBox* object.

See also *TGraphicControl.Create* method

Paint

Paint generates an *OnPaint* event.

procedure Paint; **override**;

Description *Paint* responds to WM_PAINT messages from Windows by initializing the canvas and calling the *OnPaint* event handler if it is assigned.

See also *OnPaint* event

TPanel

Unit: ExtCtrls Palette: Standard

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomControl • TCustomPanel • TPanel

TPanel implements a generic container.

Description Use *TPanel* to put an empty panel on a form. Panels provide properties for providing a bevelled border around the control, as well as methods to help manage the placement of child controls embedded in the panel.

TPanel implements the generic behavior introduced in *TCustomPanel*. *TPanel* publishes many of the properties inherited from *TCustomPanel*, but does not introduce any new behavior.

See also *TDBNavigator* object, *TToolBar* object

Properties

The following table lists the properties of *TPanel*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
Alignment	published	taCenter	TCustomPanel
BevelInner	published	bvNone	TCustomPanel
BevelOuter	published	bvRaised	TCustomPanel
BevelWidth	published	1	TCustomPanel
BorderStyle	published	bsNone	TCustomPanel
BorderWidth	published	0	TCustomPanel
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Caption	published		TControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clBtnFace	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
Locked	published	False	TCustomPanel

Property	Visibility	Default	Documented in
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Events

The following table lists the events of *TPanel*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnResize	published	TCustomPanel
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TPanel*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomPanel
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TCustomControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent

Method	Visibility	Documented in
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

TPasswordDialog

Unit: DBPWDlg

TObject • TPersistent • TComponent • TControl • TWinControl • TScrollingWinControl • TCustomForm • TForm • TPasswordDialog

TPasswordDialog is a dialog box that allows the user to specify passwords for the current session.

Description Use *TPasswordDialog* to allow the user to specify passwords for the current session. (The current session is represented by a *TSession* object.) The passwords are used when the application opens a Paradox table that requires a password. If a table requires a password that has not been added to the current session, the user will be prompted for the password.

A *TPasswordDialog* object is usually created with the *PasswordDialog* function. However, *TPasswordDialog* can also be used as a base object for other types of password dialog boxes.

See also *PasswordDialog* function, *TLoginDialog* object, *TSession* object

Properties

The following table lists the properties of *TPasswordDialog*:

Property	Visibility	Default	Documented in
Active	public, read-only		TCustomForm
ActiveControl	published		TCustomForm
ActiveMDIChild	public, read-only		TCustomForm
ActiveOleControl	public		TCustomForm
Align	public	alNone	TControl
AutoScroll	published	True	TScrollingWinControl
BorderIcons	published	[biSystemMenu, biMinimize, biMaximize]	TCustomForm
BorderStyle	published	bsSizeable	TCustomForm
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Canvas	public, read-only		TCustomForm
Caption	published		TControl
ClientHandle	public, read-only		TCustomForm
ClientHeight	published		TCustomForm
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TCustomForm
ClientWidth	published		TCustomForm
Color	published	clWindow	TControl

P

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published	True	TWinControl
Cursor	published	crDefault	TControl
Designer	public		TCustomForm
DesignInfo	public		TComponent
DropTarget	public		TCustomForm
Enabled	published	True	TControl
Font	published		TControl
FormStyle	published	fsNormal	TCustomForm
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
HorzScrollBar	published		TScrollingWinControl
Icon	published		TCustomForm
KeyPreview	published	False	TCustomForm
Left	published		TControl
MDIChildCount	public, read-only		TCustomForm
MDIChildren	public, read-only		TCustomForm
Menu	published		TCustomForm
ModalResult	public		TCustomForm
Name	published		TControl
ObjectMenuItem	published		TCustomForm
OleFormObject	public		TCustomForm
Owner	public, read-only		TComponent
Parent	public		TCustomForm
ParentFont	published	False	TControl
ParentWindow	public		TWinControl
PixelsPerInch	published		TCustomForm
PopupMenu	published		TControl
Position	published	poDesigned	TCustomForm
PrintScale	published	poProportional	TCustomForm
Scaled	published	True	TCustomForm

Property	Visibility	Default	Documented in
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	public	–1	TWinControl
TabStop	public	False	TWinControl
Tag	published	0	TComponent
TileMode	public	tbHorizontal	TCustomForm
Top	published		TControl
VCLComObject	public		TComponent
VertScrollBar	published		TScrollingWinControl
Visible	published	False	TCustomForm
Width	published		TControl
WindowMenu	published		TCustomForm
WindowProc	public		TControl
WindowState	published	wsNormal	TCustomForm

Events

The following table lists the events of *TPasswordDialog*:

Event	Visibility	Documented in
OnActivate	published	TCustomForm
OnClick	published	TControl
OnClose	published	TCustomForm
OnCloseQuery	published	TCustomForm
OnCreate	published	TCustomForm
OnDblClick	published	TControl
OnDeactivate	published	TCustomForm
OnDestroy	published	TCustomForm
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnHide	published	TCustomForm
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnPaint	published	TCustomForm
OnResize	published	TCustomForm
OnShow	published	TCustomForm

Methods

The following table lists the methods of *TPasswordDialog*:

Method	Visibility	Documented in
AddButtonClick	public	TPasswordDialog
ArrangeIcons	public	TForm
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
Cascade	public	TForm
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
Close	public	TCustomForm
CloseQuery	public	TCustomForm
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomForm
CreateNew	public	TCustomForm
CreateParented	public	TWinControl
DefaultHandler	public	TObject
DefocusControl	public	TCustomForm
Destroy	public	TCustomForm
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
DisableAutoRange	public	TScrollingWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EditChange	public	TPasswordDialog
EnableAlign	public	TWinControl
EnableAutoRange	public	TScrollingWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent

Method	Visibility	Documented in
FocusControl	public	TCustomForm
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetFormImage	public	TCustomForm
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TCustomForm
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Next	public	TForm
OKButtonClick	public	TPasswordDialog
PaintTo	public	TWinControl
Perform	public	TControl
Previous	public	TForm
Print	public	TCustomForm
Realign	public	TWinControl
Refresh	public	TControl
Release	public	TCustomForm
RemoveAllButtonClick	public	TPasswordDialog
RemoveButtonClick	public	TPasswordDialog
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl

Method	Visibility	Documented in
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
ScrollInView	public	TScrollingWinControl
SendCancelMode	public	TCustomForm
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TCustomForm
SetFocusedControl	public	TCustomForm
SetTextBuf	public	TControl
Show	public	TCustomForm
ShowModal	public	TCustomForm
Tile	public	TForm
Update	public	TWinControl
UpdateControlState	public	TWinControl

AddButtonClick

AddButtonClick controls what happens when the Add button is clicked.

```
procedure AddButtonClick(Sender: TObject);
```

Description *AddButtonClick* adds the password in the edit box to the current session by calling the *TSession.AddPassword* method. Then *AddButtonClick* clears and sets focus to the edit box.

After the *AddButtonClick* method is called, the *OKButton* is enabled, regardless of whether the edit box currently contains a password.

See Also *OKButtonClick* method, *TSession.AddPassword* method

EditChange

EditChange controls what happens when the contents of the edit box change.

```
procedure EditChange(Sender: TObject);
```

Description *EditChange* enables or disables the Add, OK, and Remove buttons according to whether the edit box contains a password. If the edit box contains a password, the buttons are enabled. Otherwise the buttons are disabled.

Note If the *AddButtonClick* method has been called, then the OK button is enabled, regardless of whether the edit box contains a password.

See Also *AddButtonClick* method

OKButtonClick

OKButtonClick controls what happens when the OK button is clicked.

```
procedure OKButtonClick(Sender: TObject);
```

Description *OKButtonClick* adds the password in the edit box to the current session by calling the *TSession.AddPassword* method.

See Also *TSession.AddPassword* method

RemoveAllButtonClick

RemoveAllButtonClick removes all passwords from the current session.

```
procedure RemoveAllButtonClick(Sender: TObject);
```

Description *RemoveAllButtonClick* removes all passwords from the current session by calling the *TSession.RemoveAllPasswords* method. Then *RemoveAllButtonClick* sets focus to the edit box.

See Also *RemoveButtonClick* method, *TSession.RemoveAllPasswords* method

RemoveButtonClick

RemoveButtonClick removes the password in the edit box from the current session.

```
procedure RemoveButtonClick(Sender: TObject);
```

Description *RemoveButtonClick* removes the password in the edit box from the current session by calling the *TSession.RemovePassword* method. Then *RemoveButtonClick* clears and sets focus to the edit box.

See Also *RemoveAllButtonClick* method, *TSession.RemovePassword* method

TPersistent

Unit: Classes

TObject • TPersistent

TPersistent is the ancestor for all objects in Delphi that have assignment and streaming capabilites.

Description *TPersistent* encapsulates the fundamental behavior common to all objects that can be assigned to another object, and that can read and write their properties to and from a stream. For this purpose *TPersisent* introduces methods that can be overridden to

- Define the procedure for loading and storing unpublished data to a stream.

- Provide the means to assign values to properties.
- Provide the means to assign the contents of one object to another.

Do not create instances of *TPersistent*. Use *TPersistent* as a base class when declaring objects that are not components, but that need streaming or assignment capabilities.

See Also *TComponent* object, *TFiler* object, *TReader* object, *TStream* object, *TWriter* object

Methods

The following table lists the methods of *TPersistent*:

Method	Visibility	Documented in
Assign	public	TPersistent
AssignTo	protected	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TObject
DefaultHandler	public	TObject
DefineProperties	protected	TPersistent
Destroy	public	TPersistent
Dispatch	public	TObject
FieldAddress	public	TObject
Free	public	TObject
FreeInstance	public	TObject
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetOwner	protected	TPersistent
InheritsFrom	public	TObject
InitInstance	public	TObject
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
SafeCallException	public	TObject

Assign

Assign copies one object to another by copying the contents of that object to the other.

procedure Assign(Source: TPersistent);

Description Override *Assign* to copy data from one object to another object of the same type. A small number of Delphi classes also support assignment to objects of different types; in all other cases, if two objects are descended from *TPersistent* but are not of the same type, the call to *Assign* raises an *EConvertError* exception.

The standard form of a call to *Assign* is

```
Destination.Assign(Source);
```

which tells the *Destination* object to copy the contents of the *Source* object to itself. In general, the statement "Destination := Source" is *not* the same as the statement "Destination.Assign(Source)". The statement "Destination := Source" makes *Destination* reference the same object as *Source*, whereas "Destination.Assign(Source)" copies the contents of the object referenced by *Source* into the different object referenced by *Destination*.

Note The types of some properties are also objects. If these properties have write methods that use *Assign* to set the value of the property, then in these cases the statement "Destination := Source" is the same as "Destination.Assign(Source)".

Conversion can happen in either the *Assign* method or the *AssignTo* method. The difference is in who does the work to implement the assignment.

Note also that *Source* can be **nil**.

See also *AssignTo* method

AssignTo

AssignTo provides a generic mechanism for assigning the data associated with an object to another object.

procedure AssignTo(Dest: TPersistent); **virtual**;

Description *AssignTo* is the protected implementation of the public *Assign* method, By default, the *AssignTo* method defined by *TPersistent* raises an *EConvertError* exception. Descendant classes can define an *AssignTo* method that copies the contents of an object of its class into another object. AssignTo exists so that knowledge of how to copy data between types does not have to exist in both participants. For example, given the following code in which *A* and *B* are instance variables:

```
A.Assign(B);
```

if *A* knows how to handle *B*, then it does so and returns. If *A* doesn't know how to handle *B*'s type, execution will trickle up to *TPersistent.Assign*, which will call:

```
B.AssignTo(A);
```

If *B* doesn't know about *A*, then an exception is raised.

See also *Assign* method

DefineProperties

The *DefineProperties* method defined by *TPersistent* does nothing other than provide an interface for a methods that read and write otherwise unpublished data.

procedure DefineProperties(Filer: TFiler); **virtual**;

Description Descendant objects of *TPersistent* can override *DefineProperties* to designate a method for storing the object's unpublished data on a stream such as a form file. By default, writing an object to a stream writes the values of all its published properties, and reading the object in reads those values and assigns them to the properties. Objects can also specify methods that read and write data other than published properties by overriding the *DefineProperties* method.

When overriding *DefineProperties*, consider including some or all of the following:

- A call to the inherited method
- Calls to the filer object's *DefineProperty* method
- Calls to the filer object's *DefineBinaryProperty* method

DefineProperties is virtual, so descendant classes can override it as necessary but are not required to redefine it.

See also *TComponent.DefineProperties* method, *TFiler.DefineBinaryProperty* method, *TFiler.DefineProperty* method

Destroy

Destroy disposes of an instance of a persistent object.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, an application should call *Free*. *Free* verifies that the persistent object has not already been freed before it calls *Destroy*.

Destroy handles cleaning up references in the global fixup list for Form Linking. Then it calls the inherited *Destroy*.

See also *Free* method, *TObject.Destroy* method

GetNamePath

GetNamePath returns a string used by the Object Inspector.

```
function GetNamePath: string; dynamic;
```

Description *GetNamePath* is for internal use only. It is used to determine the text to display in the Object Inspector for the name of the object being edited. *GetNamePath* is introduced in *TPersistent* so that descendants like collections can appear in the Object Inspector. Do not call *GetNamePath* directly.

Note For components, this is the component name. For *TCollectionItem* objects it is the name of the hosting component, the name of the property, and the index into the collection surrounded by brackets.

See also *GetOwner* method, *TComponent.DefineProperties* method, *TFiler.DefineBinaryProperty* method, *TFiler.DefineProperty* method

GetOwner

GetOwner returns the owner of a component.

```
function GetOwner: TPersistent; dynamic;
```

Description *GetOwner* is used by the *GetNamePath* method to find the owner of a component. *GetNamePath* and *GetOwner* are introduced in *TPersistent* so that descendants like collections can appear in the Object Inspector. By default for *TPersistent GetOwner* returns **nil**. For *TCollection*, it is the owner of the collection. The *TCollectionItem.GetOwner* returns the collection object the item into which the collection object has been inserted. For *TComponent* it is the value of the *Owner* property.

See also *GetPathName* method, *TCollection* object, *TComponent* object, *TComponent.Owner* property

TPopupMenu

Unit: Menus | Palette: Standard

TObject • TPersistent • TComponent • TMenu • TPopupMenu

TPopupMenu encapsulates the properties, methods, and events of a pop-up menu.

Description Use *TPopupMenu* to define the pop-up menu that appears when the user clicks on a control with the right mouse button. To make a pop-up menu available, assign the *TPopupMenu* object to the control's *PopupMenu* property.

See also *TMainMenu* object, *TMenuItem* object

Properties

The following table lists the properties of *TPopupMenu*:

Property	Visibility	Default	Documented in
Alignment	published	paLeft	TPopupMenu
AutoPopup	published	True	TPopupMenu
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DesignInfo	public		TComponent
Handle	public, read-only		TMenu
HelpContext	published	0	TPopupMenu
Items	published, read-only		TMenu
Name	published		TComponent
Owner	public, read-only		TComponent
PopupComponent	public		TPopupMenu
Tag	published	0	TComponent
VCLComObject	public		TComponent
WindowHandle	public		TMenu

Alignment

Alignment determines where the pop-up menu appears when the user clicks the right mouse button.

```
property Alignment: TPopupAlignment;
```

Description Set *Alignment* to control the placement of the popup menu. These are the possible values:

Value	Meaning
paLeft	The pop-up menu appears with its top left corner under the mouse pointer.
paCenter	The pop-up menu appears with the top center of the menu under the mouse pointer.
paRight	The pop-up menu appears with its top right corner under the mouse pointer.

The default value is *paLeft*.

See also *AutoPopup* property, *OnPopup* event

AutoPopup

AutoPopup determines whether the pop-up menu appears automatically when the user clicks the right mouse button.

```
property AutoPopup: Boolean;
```

Description Set *AutoPopup* to *True* to cause the pop-up menu to appear automatically when the user clicks the right mouse button on a control that has the pop-up menu specified as its *PopupMenu* property. The application key present on keyboards specifically designed for Windows 95 will also show this menu when pressed.

Set *AutoPopup* to *False* if the appearance of the pop-up menu should be controlled programmatically. To display a pop-up menu when *AutoPopup* is *False*, use the *Popup* method.

See also *Popup* method, *OnPopup* event, *TControl.PopupMenu* property

HelpContext

HelpContext is a context ID for the entire pop-up menu.

```
type THelpContext = -MaxLongint..MaxLongint;
```

```
property HelpContext: THelpContext;
```

Description Use *HelpContext* to associate a help screen with the entire pop-up menu. Each screen in the Help system should have a unique context ID. When a component is selected in the application, pressing F1 displays the Help screen associated with the value of *HelpContext*.

To associate a help screen with the individual items in the pop-up menu, use the *HelpContext* property of the associated *TMenuItem* objects.

See also *GetHelpContext* method, *TApplication.HelpFile* property, *TMenuItem.HelpContext* property

PopupComponent

PopupComponent is the component that last displayed the popup menu in response to a right mouse click.

```
property PopupComponent: TComponent;
```

Description Read *PopupComponent* to determine which control is currently using the popup menu. In applications where multiple controls share the same pop-up menu, use *PopupComponent* to determine which of them displayed the menu.

Set *PopupComponent* to associate a control with the menu before calling the *Popup* method programmatically to bring up the pop-up menu.

See also *AutoPopup* property, *OnPopup* event, *Popup* method, *TControl.PopupMenu* property

Events

The following table lists the events of *TPopupMenu*:

Event	Visibility	Documented in
OnPopup	published	TPopupMenu

OnPopup

OnPopup occurs just before the pop-up menu appears.

```
type TNotifyEvent = procedure (Sender: TObject) of object;

property OnPopup: TNotifyEvent;
```

Description Write an *OnPopup* event handler to take specific action just before the pop-up menu appears. For example, use an *OnPopup* event handler to set the *Checked*, *Enabled*, or *Visible* property of individual items in the menu so that they are appropriate to the *PopupComponent*.

See also *AutoPopup* property, *Popup* method, *PopupComponent* property, *TControl.PopupMenu* property

Methods

The following table lists the methods of *TPopupMenu*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TPopupMenu
DefaultHandler	public	TObject
Destroy	public	TPopupMenu
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
DispatchCommand	public	TMenu
DispatchPopup	public	TMenu
FieldAddress	public	TObject
FindComponent	public	TComponent

Method	Visibility	Documented in
FindItem	public	TMenu
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetHelpContext	public	TMenu
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
IsShortCut	public	TMenu
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Popup	public	TPopupMenu
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Create

Create creates an instance of *TPopupMenu*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to create to create a pop-up menu at runtime. Pop-up menus placed in forms or data modules at design time are created automatically.

See also *TMenu.Create* method

Destroy

Destroy destroys an instance of *TPopupMenu*.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the pop-up menu is not already freed, and only then calls *Destroy*.

See also *Free* method

Popup

Popup displays the pop-up menu onscreen.

```
procedure Popup(X, Y: Integer); virtual;
```

Description Call *Popup* to bring up the pop-up menu. The menu appears at the screen coordinates indicated by the values (in pixels) of X and Y.

See also *Alignment* property, *AutoPopup* property, *OnPopup* event

TPrintDialog

Unit: Dialogs | Palette: Dialogs

TObject • TPersistent • TComponent • TCommonDialog • TPrintDialog

TPrintDialog displays a Print dialog.

Description The *TPrintDialog* component displays a standard Windows dialog box for sending jobs to a printer. The dialog is modal and does not appear at runtime until it is activated by a call to the *Execute* method.

See also *TPrinterSetupDialog* object

Properties

The following table lists the properties of *TPrintDialog*:

Property	Visibility	Default	Documented in
Collate	published	False	TPrintDialog
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Copies	published	0	TPrintDialog
Ctl3D	published	True	TCommonDialog
DesignInfo	public		TComponent
FromPage	published	0	TPrintDialog
HelpContext	published	0	TCommonDialog
MaxPage	published	0	TPrintDialog
MinPage	published	0	TPrintDialog
Name	published		TComponent
Options	published	[]	TPrintDialog
Owner	public, read-only		TComponent

Property	Visibility	Default	Documented in
PrintRange	published	prAllPages	TPrintDialog
PrintToFile	published	False	TPrintDialog
Tag	published	0	TComponent
ToPage	published	0	TPrintDialog
VCLComObject	public		TComponent

Collate

Indicates whether the Collate check box is selected.

property Collate: Boolean;

Description *Collate* is *True* whenever the Collate check box is selected in the Print dialog. To make the dialog open with the check box selected, set *Collate* to *True* in the Object Inspector or in program code.

See also *Copies* property

Copies

Indicates the number copies selected in the Print dialog.

property Copies: Integer;

If *Copies* is 0 or 1, the dialog will have 1 in the Number of Copies field. To make the dialog open with 2 or more copies selected, set the value of *Copies* in the Object Inspector or in program code.

See also *Collate* property

P

FromPage

Indicates the page on which the print job is to begin.

property FromPage: Integer;

Description The *FromPage* property corresponds to the From field in the Print dialog.

See also *MaxPage* property, *MinPage* property, *Options* property, *PrintRange* property, *ToPage* property

MaxPage

Determines the greatest page number the user can enter when specifying pages to print.

property MaxPage: Integer;

Description If the user tries to enter a number greater than *MaxPage*, an error message appears.

Note The user can specify page numbers only if the *poPageNums* flag is set in *Options*. If *poPageNums* is on, *MaxPage* must be set to a value greater than *MinPage* or the Print dialog will not open.

See also *FromPage* property, *MinPage* property, *Options* property, *ToPage* property

MinPage

Determines the smallest page number the user can enter when specifying pages to print.

```
property MinPage: Integer;
```

Description If the user tries to enter a number smaller than *MinPage*, a error message appears.

Note The user can specify page numbers only if the *poPageNums* flag is set in *Options*. If *poPageNums* is on, *MaxPage* must be set to a value greater than *MinPage* or the Print dialog will not open.

See also *FromPage* property, *MaxPage* property, *Options* property, *ToPage* property

Options

Determines the appearance and behavior of the Print dialog.

```
type
  TPrintDialogOption = (poPrintToFile, poPageNums, poSelection, poWarning, poHelp,
    poDisablePrintToFile);
  TPrintDialogOptions = set of TPrintDialogOption;

property Options: TPrintDialogOptions;
```

Description The possible values of *Options* are:

Value	Meaning
poDisablePrintToFile	Disables (grays) the Print To File Check box. (Applicable only if the *poPrintToFile* flag is set.)
poHelp	Displays a Help button in the dialog. May not work in some versions of Windows 95.
poPageNums	Enables the Pages radio button, allowing the user to specify a page range.
poPrintToFile	Displays a Print To File check box in the dialog.
poSelection	Enables the Selection radio button, allowing the user to print selected (highlighted) text.
poWarning	Generates a warning message if the user tries to send a job to an uninstalled printer.

By default, all options flags are off.

See also *FromPage* property, *MaxPage* property, *MinPage* property, *ToPage* property

PrintRange

Indicates the type of print range selected in the dialog.

```
type TPrintRange = (prAllPages, prSelection, prPageNums);
property PrintRange: TPrintRange;
```

Description The value of *PrintRange* corresponds the All, Selection, and Pages (From/To) radio buttons in the Print dialog:

Value	Meaning
prAllPages	The All radio button is selected.
prSelection	The Selection radio button is selected.
prPageNums	The Pages radio button is selected.

To determine which radio button is selected by default when the dialog opens, set the value of *PrintRange* in the Object Inspector or in program code.

Note To enable the Pages radio button, set the *poPageNums* flag in *Options*. To enable the Selection radio button, set the *poSelection* flag in *Options* and set *MaxPage* to a value greater than *MinPage*.

See also *MaxPage* property, *MinPage* property, *Options* property

PrintToFile

Indicates whether the Print To File check box is selected.

```
property PrintToFile: Boolean;
```

Description *PrintToFile* is *True* whenever the Print To File check box is selected in the dialog. To make the dialog open with the check box selected, set *PrintToFile* to *True* in the Object Inspector or in program code.

Note To make the Print To File check box appear in the dialog, set the *poPrintToFile* flag in *Options*. To disable (gray) the Print To File check box, set the *poDisablePrintToFile* flag.

See also *Options* property

ToPage

Indicates the page on which the print job is to end.

```
property ToPage: Integer;
```

Description The *ToPage* property corresponds to the To field in the Print dialog.

See also *FromPage* property, *MaxPage* property, *MinPage* property

Events

The following table lists the events of *TPrintDialog*:

Event	Visibility	Documented in
OnClose	published	TCommonDialog
OnShow	published	TCommonDialog

Methods

The following table lists the methods of *TPrintDialog*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TCommonDialog
DefaultHandler	public	TObject
Destroy	public	TCommonDialog
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
Execute	public	TPrintDialog
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject

Method	Visibility	Documented in
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Execute

Displays the Print dialog box.

function Execute: Boolean; **override**;

Description *Execute* opens the Print dialog, returning *True* when the user clicks Print and *False* when the user clicks Cancel.

TPrinterSetupDialog

Unit: Dialogs Palette: Dialogs

TObject • TPersistent • TComponent • TCommonDialog • TPrinterSetupDialog

TPrinterSetupDialog displays a Print Setup dialog for configuring printers.

Description *TPrinterSetupDialog* displays a modal Windows dialog box for configuring printers. The contents of the dialog vary depending on the printer driver selected.

The dialog does not appear at runtime until it is activated by a call to the *Execute* method.

See also *TPrintDialog* object

Properties

The following table lists the properties of *TPrinterSetupDialog*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Ctl3D	published	True	TCommonDialog
DesignInfo	public		TComponent

Property	Visibility	Default	Documented in
HelpContext	published	0	TCommonDialog
Name	published		TComponent
Owner	public, read-only		TComponent
Tag	published	0	TComponent
VCLComObject	public		TComponent

Events

The following table lists the events of *TPrinterSetupDialog*:

Event	Visibility	Documented in
OnClose	published	TCommonDialog
OnShow	published	TCommonDialog

Methods

The following table lists the methods of *TPrinterSetupDialog*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TCommonDialog
DefaultHandler	public	TObject
Destroy	public	TCommonDialog
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
Execute	public	TPrinterSetupDialog
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject

Method	Visibility	Documented in
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Execute

Displays the printer configuration dialog box.

function Execute: Boolean; **override**;

Description *Execute* opens the Print Setup dialog, returning *True* when the user clicks OK.

TProgressBar

Unit: ComCtrls Palette: Win32

TObject • TPersistent • TComponent • TControl • TWinControl • TProgressBar

TProgressBar is a wrapper for a Windows progress bar.

Description Use *TProgressBar* to add a progress bar to a form. Progress bars provide users with visual feedback about the progress of a procedure within an application. As the procedure progresses, the rectangular progress bar gradually fills from left to right with the system highlight color.

Properties

The following table lists the properties for *TProgressBar*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl

Property	Visibility	Default	Documented in
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
Max	published		TProgressBar
Min	published		TProgressBar
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
Position	published	0	TProgressBar
ShowHint	published		TControl
Showing	public, read-only		TWinControl
Step	published	10	TProgressBar
TabOrder	published	–1	TWinControl
TabStop	published	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl

Property	Visibility	Default	Documented in
Width	published		TControl
WindowProc	public		TControl

Max

Max specifies the upper limit of the range of possible positions.

property Max: Integer;

Description Use *Max* along with the *Min* property to establish the range of possible positions a progress bar. When the process tracked by the progress bar is complete, the value of *Position* should equal *Max*.

See also *Max* property, *Position* property

Min

Min specifies the lower limit of the range of possible positions.

property Min: Integer;

Description Use *Max* along with the *Min* property to establish the range of possible positions a progress bar. When the process tracked by the progress bar begins, the value of *Position* should equal *Min*.

See also *Min* property, *Position* property

Position

Position is the current position of the progress bar.

property Position: Integer;

Description Read *Position* to determine how far the process tracked by the progress bar has advanced from *Min* toward *Max*. Set *Position* to cause the progress bar to display a position between *Min* and *Max*. For example, when the process tracked by the progress bar completes, set *Position* to *Max* so that it appears completely filled.

When a progress bar is created, *Min* and *Max* represent percentages, where *Min* is 0 (0% complete) and *Max* is 100 (100% complete). If these values are not changed, *Position* is the percentage of the process that has already been completed.

See also *Max* property, *Min* property, *StepBy* method, *StepIt* method

Step

Step is the amount that *Position* increases when the *StepIt* method is called.

property Step: Integer;

Description Set *Step* to specify the granularity of the progress bar. *Step* should reflect the size of each step in the process tracked by the progress bar, in the logical units used by the *Max* and *Min* properties.

When a progress bar is created, *Min* and *Max* represent percentages, where *Min* is 0 (0% complete) and *Max* is 100 (100% complete). If these values are not changed, *Step* is the percentage of the process completed before the user is provided with additional visual feedback.

When the *StepIt* method is called, the value of *Position* increases by *Step*.

See also *Max* property, *Min* property, *Position* property, *StepIt* method

Events

The following table lists the events for *TProgressBar*:

Event	Visibility	Documented in
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods for *TProgressBar*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl

Method	Visibility	Documented in
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TProgressBar
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TWinControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject

Method	Visibility	Documented in
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
StepBy	public	TProgressBar
StepIt	public	TProgressBar
Update	public	TWinControl
UpdateControlState	public	TWinControl

Create

Create creates an instance of *TProgressBar*.

```
constructor Create(AOwner: TComponent); override;
```

Description Call *Create* to create a progress bar at runtime. Progress bars placed on forms at design time are created automatically. After calling the inherited constructor, *Create* initializes the following values:

- *Min* is set to 0 and *Max* to 100.
- *Step* is set to 10.
- *Width* is set to 150.
- *Height* is set to the height the arrow bitmap on a scrollbar.

See also *Height* property, *Max* property, *Min* property, *Step* property, *TWinControl.Create* method, *Width* property

StepBy

StepBy advances the *Position* of the progress bar by a specified amount.

```
procedure StepBy(Delta: Integer);
```

Description Call *StepBy* to increase the value of *Position* by the value of the *Delta* parameter. To advance *Position* by a default amount that represents a single step in the process, use the *StepIt* method.

See also *Position* property, *StepIt* method

StepIt

StepIt advances *Position* by the amount specified in the *Step* property.

```
procedure StepIt;
```

Description Call the *StepIt* method to increase the value of *Position* by the value of the *Step* property. If *Step* represents the size of one logical step in the process tracked by the progress bar, call *Step* after each logical step is completed.

See also *Position* property, *Step* property, *StepBy* method

TPropertyPage

Unit: AxCtrls

TObject • TPersistent • TComponent • TControl • TWinControl • TScrollingWinControl • TCustomForm • TPropertyPage

TPropertyPage is an abstract class that implements a Delphi form that can be used as a property page for an ActiveX control.

Description *TPropertyPage* extends *TForm* by adding the functionality needed to embed the form in an ActiveX property page dialog. It also adds methods to copy property values from the object to the form's controls, and vice versa.

To implement your own property page for an ActiveX object, derive a new class from *TPropertyPage*, design the form, then override the *UpdateObject* and *UpdatePropertyPage* methods. The Delphi IDE provides an expert for creating a skeleton property page. See the *Programmer's Guide* for more information.

TPropertyPage provides a helper method, *Modified*, that your form can call whenever any of the controls on the form are changed by the user. Calling the *Modified* method will notify the property dialog that the user has modified the object's properties. Normally this will enable the Apply button on the property dialog.

The property page maintains a connection to the object it is editing, using the *OleObject* property. This property is a *Variant* and can be used to read and write the object's properties.

See also *TActiveXPropertyPageFactory* object

Properties

The following table lists the properties of *TPropertyPage*:

Property	Visibility	Default	Documented in
Active	public, read-only		TCustomForm
ActiveControl	published		TCustomForm
ActiveOleControl	public		TCustomForm
Align	public	alNone	TControl
AutoScroll	published	True	TScrollingWinControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Canvas	public, read-only		TCustomForm
Caption	published		TControl
ClientHeight	published		TCustomForm
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TCustomForm
ClientWidth	published		TCustomForm
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published	True	TWinControl
Cursor	published	crDefault	TControl
Designer	public		TCustomForm
DesignInfo	public		TComponent
DropTarget	public		TCustomForm
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
HorzScrollBar	published		TScrollingWinControl
KeyPreview	published	False	TCustomForm
Left	published		TControl

Property	Visibility	Default	Documented in
Menu	public		TCustomForm
ModalResult	public		TCustomForm
Name	published		TControl
OleFormObject	public		TCustomForm
OleObject	public, read-only		TPropertyPage
Owner	public, read-only		TComponent
Parent	public		TCustomForm
ParentFont	published	False	TControl
ParentWindow	public		TWinControl
PixelsPerInch	published		TCustomForm
PopupMenu	published		TControl
PrintScale	published	poProportional	TCustomForm
Scaled	published	True	TCustomForm
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	public	–1	TWinControl
TabStop	public	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
VertScrollBar	published		TScrollingWinControl
Visible	published	False	TCustomForm
Width	published		TControl
WindowProc	public		TControl
WindowState	public	wsNormal	TCustomForm

P

OleObject

Contains a reference to the ActiveX control being edited by the page.

property OleObject: Variant;

Description Use this property to access the property values and methods of the control.

See also *UpdatePropertyPage* method, *UpdateObject* method

Events

The following table lists the events of *TPropertyPage*:

Event	Visibility	Documented in
OnActivate	published	TCustomForm
OnClick	published	TControl

Event	Visibility	Documented in
OnClose	published	TCustomForm
OnCloseQuery	published	TCustomForm
OnCreate	published	TCustomForm
OnDblClick	published	TControl
OnDeactivate	published	TCustomForm
OnDestroy	published	TCustomForm
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnHide	published	TCustomForm
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnPaint	published	TCustomForm
OnResize	published	TCustomForm
OnShow	published	TCustomForm

Methods

The following table lists the methods of *TPropertyPage*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
Close	public	TCustomForm
CloseQuery	public	TCustomForm
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomForm

Method	Visibility	Documented in
CreateNew	public	TCustomForm
CreateParented	public	TWinControl
DefaultHandler	public	TObject
DefocusControl	public	TCustomForm
Destroy	public	TCustomForm
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
DisableAutoRange	public	TScrollingWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EnableAutoRange	public	TScrollingWinControl
EndDrag	public	TControl
EnumCtlProps	public	TPropertyPage
FieldAddress	public	TObject
FindComponent	public	TComponent
FocusControl	public	TCustomForm
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetFormImage	public	TCustomForm
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TCustomForm
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject

Method	Visibility	Documented in
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
Modified	public	TPropertyPage
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Print	public	TCustomForm
Realign	public	TWinControl
Refresh	public	TControl
Release	public	TCustomForm
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
ScrollInView	public	TScrollingWinControl
SendCancelMode	public	TCustomForm
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TCustomForm
SetFocusedControl	public	TCustomForm
SetTextBuf	public	TControl
Show	public	TCustomForm
ShowModal	public	TCustomForm
Update	public	TWinControl
UpdateControlState	public	TWinControl
UpdateObject	public	TPropertyPage
UpdatePropertyPage	public	TPropertyPage

EnumCtlProps

Looks for a *PropType* type property in a control on the form.

```
procedure EnumCtlProps(PropType: TGUID; PropNames: TStrings);
```

Description Stores the list of same type properties of a control in *PropNames*.

See also *UpdateObject* method, *UpdatePropertyPage* method

Modified

The *Modified* method is a helper method that you can call to notify the property page's container that the user has edited one of the controls on the page.

```
procedure Modified;
```

Description The property dialog will enable the Apply button in response to this method call.

See also *OnChange* event

UpdatePropertyPage

The *UpdatePropertyPage* method is called to copy the property values from the ActiveX object to controls on the form.

```
procedure UpdatePropertyPage; virtual;
```

Description This method is normally called when the property page is first shown, or when the user undoes changes to the property dialog box.

You should override this method to copy the property values from the ActiveX object to controls on your form.

UpdatePropertyPage is the reverse of *UpdateObject*, which copies the property values from controls on the form to the ActiveX object.

See also *UpdateObject* method

UpdateObject

The *UpdateObject* method is called to copy property values from the controls on the form to the ActiveX Object.

```
procedure UpdateObject; virtual;
```

Description You should override this method to copy data from the controls on the form into properties of the ActiveX Object.

UpdateObject is the reverse of *UpdatePropertyPage*, which copies the property values from the ActiveX Object to controls on the form.

See also *UpdatePropertyPage* method

TProvider

Unit: BdeProv | Palette: Data Access

TObject • TPersistent • TComponent • TCustomProvider • TProvider

Encapsulates the mechanism for providing data from a multi-tiered application server to a client dataset in a desktop client application.

Description

TProvider is an object, usually placed in a multi-tiered application server, that serves as a data broker and conduit between a remote database application server and a client dataset in multi-tiered, desktop client application. The provider offers an *IProvider* interface that is an intermediate link between the client dataset on the client, and a *TQuery*, *TStoredProc*, or *TTable* object on the application server. It packages data from the server dataset and passes it in data packets to the client dataset.

The client dataset receives the data packets, and reconstructs them to create a local, in-memory copy of the data for user access. When user access is complete, the client dataset repackages any changed data and sends the updates to the provider on the application server. The provider applies the updates to the remote database server, resolves record conflicts where possible, and sends all unresolved records back to the client dataset for further reconciliation.

See also

TClientDataSet object, *TQuery* object, *TStoredProc* object, *TTable* object

Properties

The following table lists the properties of *TProvider*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Constraints	published	True	TCustomProvider
Data	public, read-only		TProvider
DataSet	published		TProvider
DesignInfo	public		TComponent
Name	published		TComponent
Owner	public, read-only		TComponent
Provider	public, read-only		TCustomProvider
Tag	published	0	TComponent
VCLComObject	public		TComponent

Data

Retrieves data from a database server in response to client requests.

```
property Data: OleVariant;
```

Description *Data* is a read-only property that can be used to retrieve all records from a database in response to client dataset requests.

DataSet

Specifies the Borland Database Engine (BDE) dataset component on the application server from which the provider gets data and to which it posts client application updates.

```
property DataSet: TDBDataSet;
```

Description Set *DataSet* to the name of a dataset component in a data module on the application server. The specified dataset component is used by the provider to get data from the database server to which the component is connected, and to write client application updates to the database server.

DataSet must be a BDE-enabled dataset component, either one of the ones provided in the Object and Component Library with Delphi (*TTable, TStoredProc, TQuery*), or a custom component derived from *TDBDataSet*.

See also *TDBDataSet* object, *TQuery* object, *TStoredProc* object, *TTable* object

Events

P

The following table lists the events of *TProvider*:

Event	Visibility	Documented in
OnDataRequest	published	TCustomProvider
OnUpdateError	published	TProvider

OnUpdateError

OnUpdateError occurs when a provider attempts to update a record and cannot because of an error condition.

```
type
  TUpdateKind = (ukModify, ukInsert, ukDelete);
  TResolverResponse = (rrSkip, rrAbort, rrMerge, rrApply, rrIgnore);
  TResolverErrorEvent = procedure(DataSet: TClientDataSet; E: EDatabaseError;
    UpdateKind: TUpdateKind; var Response: TResolverResponse) of object;

property OnUpdateError: TResolverErrorEvent;
```

Description Write an *OnUpdateError* event handler to respond to error conditions that arise when the provider applies a client dataset's updates to the database

server. If a record cannot be updated *OnUpdateError* is called, if one is provided. The error handler can use the *NewValue*, *OldValue*, and *CurValue* properties of each field in each record to determine the cause of an error and attempt to resolve the problem.

DataSet is the name of a temporary client dataset to use for accessing data during the update process. Use the *OldValue*, *NewValue*, and *CurValue* properties of the dataset's field components to view the changed fields in each record.

E is a pointer to a *EUpdateError* object from which an application can extract an error message and the actual cause of the error condition. An *OnUpdateError* handler can use this information to determine how to respond to the error condition.

UpdateKind describes the type of update that generated the error. The following table lists the possible values for *UpdateKind* and what they indicate:

Value	Meaning
ukModify	Editing an existing record caused an error.
ukInsert	Inserting or appending a new record caused an error.
ukDelete	Deleting an existing record raised an error.

Response indicates the action to take when the *OnUpdateError* handler exits. On entry into the handler, *Response* is set to either *rrAbort* or *rrSkip*, based on the *MaxErrors* parameter passed to the *ApplyUpdates* function that triggers *OnUpdateError*. If *MaxErrors* is *0* then *Response* is *rrAbort*; otherwise it is *rrSkip*. If *OnUpdateError* can handle or correct the error, set *Response* to *rrCorrect* before exiting the error handler. The following table lists the possible values for *Response* and what they indicate:

Value	Meaning
rrSkip	Skips updating the record that raised the error condition, and leaves the unapplied changes in the cache.
rrAbort	Aborts the operation without displaying an error message.
rrMerge	Merges the changed record with the record in the database
rrCorrect	Corrects the record in the database with this record
rrApply	Applies the corrected updated to the database.
rrIgnore	Ignores the error condition and does not apply the record.

Note The code in an *OnUpdateError* handler must not call any methods that make a different record the current one.

See also *ApplyUpdates* method, *TField* object

Methods

The following table lists the methods of *TProvider*:

Method	Visibility	Documented in
ApplyUpdates	public	TProvider
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TProvider
DefaultHandler	public	TObject
Destroy	public	TComponent
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetRecords	public	TProvider
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
Reset	public	TProvider
SafeCallException	public	TComponent

ApplyUpdates

Applies updates received from a client dataset to datasets connected to a remote database server.

```
function ApplyUpdates(Delta: OleVarant; MaxErrors: Integer;
 out ErrorCount: Integer); OleVariant;
```

Description Call *ApplyUpdates* to write a client's updates to a database server. *Delta* is a variant array containing the updated, inserted, and deleted records from the client to write to the database. *MaxErrors* indicates the maximum number of errors to permit before aborting the update operation. *ErrorCount* indicates the actual number of errors encountered during the update operation. If *ErrorCount* is greater than *MaxErrors*, or the *OnUpdateError* event returns *rrAbort* for a record it cannot resolve, all updates that have been successfully applied so far are rolled back, and the update process is aborted.

If you provider an *OnUpdateError* event handler for the provider, it is called automatically whenever a record in *Delta* cannot be updated. The *OnUpdateError* handler gives you a chance to correct the error condition and try again before returning an unsuccessfully updated record back to the client dataset. Records that cannot be updated are stored in a results array that is converted and returned to the client dataset for further reconciliation after all records are processed.

ApplyUpdates returns a variant that is a data packet containing all records that could not be applied to the database.

See also *OnUpdateError* event

Create

Creates an instance of a *TProvider* component.

```
constructor Create(AOwner: TComponent);
```

Description Call *Create* to instantiate a provider component at runtime. Ordinarily, if you place a provider component in a data module for an application server at design time, it is created automatically for you at runtime.

Create calls its inherited *Create* method, points to a BDE-enabled dataset component if such a dataset owns the provider, and sets the *Constraints* property to *True*.

See also *Constraints* property, *DataSet* property, *Destroy* method

GetRecords

Retrieves a specified number of records from a provider's dataset.

```
function GetRecords(Count: Integer; out RecsOut: Integer); Variant;;
```

Description Call *GetRecords* to retrieve a specified number of records from the provider's dataset. *Count* indicates the number of records to retrieve. If *Count* is –1, all records are retrieved. If *Count* is *0*, only metadata records are retrieved. *RecsOut* indicate the actual number of records retrieved. *GetRecords* returns a variant containing a data packet.

A client application calls a provider's *GetRecords* method to retrieve a partial set of all available records from a dataset. *GetRecords* is especially useful when dealing with very large datasets, or when working with datasets that have blob fields.

Note To retrieve all records at once, call *Get_Data* instead of *GetRecords*.

See also *DataRequest* method, *Get_Data* method

Reset

Enables a provider to start fetching sequential packets of records from the start of the dataset.

```
procedure Reset;
```

Description Ordinarily, the provider fetches sequential packets of records from the database server. If the server contains 100 records, and the provider fetches 10 records at a time (based on a client's request), then the first fetch retrieves records 1 to 10, the second 11 to 20, and so on.

Call *Reset* to force a provider to start fetching packets of records from the start of the dataset again. Otherwise each fetch operation starts at the record following the last record fetched in the previous call to *GetRecords*.

See also *GetRecords* method

Q

TQuery

Unit: DBTables | Palette: Data Access

TObject • TPersistent • TComponent • TDataSet • TBDEDataset • TDBDataSet • TQuery

TQuery encapsulates a dataset with a result set that is based on an SQL statement.

Description Use *TQuery* to access a database using SQL statements. Query components can be used with remote database servers such as Sybase, SQL Server, Oracle, Informix, DB2, and InterBase, with Paradox, dBASE, Access, and FoxPro, and with ODBC-compliant databases.

Query components are useful because they can

- Access more than one table at a time (called a "join" in SQL).
- Automatically access a subset of rows and columns in its underlying table(s), rather than always returning all rows and columns.

Note *TQuery* is of particular importance to the development of scalable database applications. If there is any chance that an application built to run against local databases will be scaled to a remote SQL database server in the future, use *TQuery* components from the start to ensure easier scaling later.

See also *TStoredProc* object, *TTable* object

Properties

The following table lists the properties of *TQuery*:

Property	Visibility	Default	Documented in
Active	published	False	TDataSet
AutoCalcFields	published	True	TDataSet
BOF	public, read-only		TDataSet
Bookmark	public		TDataSet
CacheBlobs	public	True	TBDEDataSet
CachedUpdates	published	False	TBDEDataSet
CanModify	public, read-only		TBDEDataSet
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Constrained	published	False	TQuery
Constraints	published		TDataSet
Database	public, read-only		TDBDataSet
DatabaseName	published		TDBDataSet
DataSource	published		TQuery
DBHandle	public, read-only		TDBDataSet
DBLocale	public, read-only		TDBDataSet
DBSession	public, read-only		TDBDataSet
DefaultFields	public, read-only		TDataSet
Designer	public, read-only		TDataSet
DesignInfo	public		TComponent
EOF	public, read-only		TDataSet
ExpIndex	public, read-only		TBDEDataSet
FieldCount	public, read-only		TDataSet
FieldDefs	public		TDataSet
Fields	public		TDataSet
FieldValues	public		TDataSet
Filter	published		TBDEDataSet
Filtered	published	False	TBDEDataSet

Property	Visibility	Default	Documented in
FilterOptions	published	[]	TBDEDataSet
Found	public, read-only		TDataSet
Handle	public, read-only		TBDEDataSet
KeySize	public, read-only		TBDEDataSet
Local	public, read-only		TQuery
Locale	public, read-only		TBDEDataSet
Modified	public, read-only		TDataSet
Name	published		TDataSet
Owner	public, read-only		TComponent
ParamCheck	published	True	TQuery
ParamCount	public, read-only		TQuery
Params	published		TQuery
Prepared	public		TQuery
Provider	public, read-only		TDBDataSet
RecNo	public		TBDEDataSet
RecordCount	public, read-only		TBDEDataSet
RecordSize	public, read-only		TBDEDataSet
RequestLive	published	False	TQuery
RowsAffected	public, read-only		TQuery
SessionName	published		TDBDataSet
SQL	published		TQuery
SQLBinary	public		TQuery
State	public, read-only		TDataSet
StmtHandle	public, read-only		TQuery
Tag	published	0	TComponent
Text	public, read-only		TQuery
UniDirectional	published	False	TQuery
UpdateMode	published	upWhereAll	TDBDataSet
UpdateObject	published		TBDEDataSet
UpdateRecordTypes	public		TBDEDataSet
UpdatesPending	public, read-only		TBDEDataSet
VCLComObject	public		TComponent

Constrained

Indicates whether or not updates and inserts that do not conform to the result set generated by a SELECT statement are permitted for Paradox and dBASE tables

property Constrained: Boolean;

Description Use *Constrained* to determine whether or not edits and insertions are permitted to a Paradox or dBASE table underlying a result set when the UPDATE or INSERT statements contain one or more values that do not

comply with the WHERE clause conditions of the SELECT statement that generated the result set. *Constrained* only applies to live result sets. Constraints are enforced during the editing process.

If *Constrained* is *False* (the default), updates and inserts are permitted regardless of the SELECT statement criteria. If *True*, updates and inserts are not permitted if they violate the criteria of the SELECT statement that generated the result set.

For example, if *Constrained* is *True* and a query component's *SQL* property is

```
SELECT * FROM CUSTOMERS
  WHERE (CUSTNO > 1400 AND CUSTNO < 1500)
```

then the user would not be allowed to enter a value for *CustNo* that fell outside this range of values.

See also *SQL* property

DataSource

Specifies the data source component from which to extract current field values to use to bind otherwise unassigned parameters in the query's SQL statement.

property DataSource: TDataSource;

Description Set the *DataSource* property to the name of a data source component from which to extract current field values to use to bind otherwise unassigned parameters in the SQL statement in this component's *SQL* property. *DataSource* must point to another dataset; it cannot point to this query component.

The dataset specified in *DataSource* must be created, populated, and opened before attempting to bind parameters. Parameters are bound by calling the query's *Prepare* method prior to executing the query.

Tip *DataSource* is especially of use when creating a master-detail relationship between tables using a linked query. It is also of use to guarantee binding for parameters that are not already set in the *Params* property or through a call to the *ParamByName* method.

If the SQL statement used by a query does not contain parameters, or all parameters are bound by the application using the Params property or the ParamByName method, DataSource need not be assigned.

See also *ParamByName* method, *Params* property, *SQL* property

Local

Indicates whether a query references one or more Paradox or dBASE tables rather than SQL tables on a remote server.

property Local: Boolean;

Description Check *Local* to determine if a query accesses one or more Paradox or dBASE tables rather than SQL tables. If *Local* is *True*, then the query only accesses dBASE or Paradox. Otherwise *Local* is *False*. *Local* is set automatically when an application calls the query's *Prepare* method prior to executing the query.

Some query component settings, such as *Constrained*, are only meaningful for queries against Paradox and dBASE tables. Others are only meaningful for queries against remote SQL servers. *Local* can provide useful information for branching logic in code based on the type of database accessed. It can also be used to determine whether or not the BDE parsed a query using its local query engine rather than passing SQL statements directly to a remote server.

See also *Constrained* property, *ExecSQL* method, *Open* method, *Prepare* method

ParamCheck

Specifies whether the parameter list for a query is regenerated if the SQL property changes at runtime.

property ParamCheck: Boolean;

Description Set *ParamCheck* to specify whether or not the *Params* property is cleared and regenerated if an application modifies the query's *SQL* property at runtime. By default *ParamCheck* is *True*, meaning that the *Params* property is automatically regenerated at runtime. When *ParamCheck* is *True*, the proper number of parameters is guaranteed to be generated for the current SQL statement.

This property is most useful for data definition statements that use parameters (for example, when using a query component to create an InterBase stored procedure).

An application that does not use parameterized queries may choose to set *ParamCheck* to *False*, but otherwise *ParamCheck* should be *True*.

See also *Params* property, *SQL* property, TStoredProc object

ParamCount

Indicates the current number of parameters for the query.

property ParamCount: Word;

Description Check *ParamCount* to determine how many parameters are in the *Params* property. If the *ParamCheck* property is *True*, *ParamCount* always corresponds to the number of actual parameters in the SQL statement for the query.

An application can add or delete parameters to the *Params* property. Such additions and deletions are automatically reflected in *ParamCount*.

See also *Params* property, *SQL* property

Params

Contains the parameters for a query's SQL statement.

property Params[Index: Word]: TParams;

Description Access *Params* at runtime to view and set parameter names, values, and data types dynamically (at design time use the Query Parameters editor to set parameter information). *Params* is a zero-based array of parameter records. *Index* specifies the array element to access.

Note An easier way to set and retrieve parameter values when the name of each parameter is known is to call *ParamByName*. *ParamByName* cannot, however, be used to change a parameter's data type or name.

See also *ParamByName* method, *SQL* property

Prepared

Determines whether or not a query is prepared for execution.

property Prepared: Boolean;

Description Examine *Prepared* to determine if a query is already prepared for execution. If *Prepared* is *True*, the query is prepared, and if *Prepared* is *False*, the query is not prepared. While a query need not be prepared before execution, execution performance is enhanced if the query is prepared beforehand, particularly if it is a parameterized query that is executed more than once using the same parameter values.

Note An application can change the current setting of *Prepared* to prepare or unprepare a query. If *Prepared* is *True*, setting it to *False* calls the *Unprepare* method to unprepare the query. If *Prepared* is *False*, setting it to *True* calls the *Prepare* method to prepare the query. Generally, however, it is better programming practice to call *Prepare* and *Unprepare* directly. These methods automatically update the *Prepared* property.

See also *ExecSQL* method, *Open* method, *Prepare* method, *Unprepare* method

RequestLive

Specifies whether an application expects to receive a live result set from the Borland Database Engine (BDE) when the query executes.

property RequestLive: Boolean;

Description Set *RequestLive* to specify whether or not the BDE should attempt to return a live result set to the application. *RequestLive* is *False* by default, meaning that a query returns a read-only result set.

Set *RequestLive* to *True* to request a live result set. Setting *RequestLive* to *True* does not guarantee that a live result set is returned by the BDE. It only guarantees that the BDE attempts to return a live result set. The BDE returns

a live result set only if the SELECT syntax of the query conforms to the syntax requirements for a live result set.

If *RequestLive* is *True*, but the syntax does not conform to the requirements, the BDE returns a read-only result set for Paradox or dBASE, or an error return code for remote servers.

See also *CanModify* property, *ExecSQL* property, *Open* property, *SQL* property

RowsAffected

Returns the number of rows operated upon by the latest query execution.

property RowsAffected: Integer;

Description Check *RowsAffected* to determine how many rows were updated or deleted by the last query operation. If *RowsAffected* is –1, the query did not update or delete any rows.

See also *ExecSQL* method

SQL

Contains the text of the SQL statement to execute for the query.

property SQL: TStrings;

Description Use *SQL* to provide the SQL statement that a query component executes when its *ExecSQL* or *Open* methods are called. At design time the SQL property can be edited by invoking the String List editor in the Object Inspector.

Note Delphi Client/Server developers can also invoke the Visual Query Builder to create an SQL statement for execution.

The SQL property can be used to access

- Paradox or dBASE tables, using local SQL. The allowable syntax is a subset of ANSI-standard SQL and includes SELECT, INSERT, UPDATE, and DELETE statements.
- Databases on the Local InterBase Server, using standard SQL-92 and specific InterBase extensions to SQL syntax. For information on syntax and limitations, see the InterBase *Language Reference*.
- Databases on remote database servers (Delphi Client/Server only) using the appropriate SQL Links driver. Any standard statement in the server's SQL dialect is allowed. For information on SQL syntax and limitations, see the server documentation.

The SQL statement provided to *SQL* may contain replaceable parameters, following standard SQL-92 syntax conventions. Parameters are created and stored in the *Params* property.

To provide an SQL statement at runtime:

1. Call *Close* to shut down the query if it is currently active.
2. Call *Clear* to empty the *SQL* property.
3. Call *Add* to specify the new text, or call *LoadFromFile* to assign the text in an SQL script file to the *SQL* property.
4. Use the *ParamByName* method to set up any parameters for the query, if it uses them.
5. Call *Prepare* to ready the query for execution.
6. Call *ExecSQL* or *Open* to execute the query.

Note The *SQL* property may contain only one complete SQL statement at a time. In general, multiple "batch" statements are not allowed unless a particular server supports them.

See also *Add* method, *Clear* method, *Close* method, *ExecSQL* method, *LoadFromFile* method, *Open* method, *ParamByName* method, *Prepare* method

SQLBinary

Points to the binary data stream that represents an SQL query statement or result set.

```
property SQLBinary: PChar;
```

Description Do not access *SQLBinary*. It is an internal binary data stream used by the query component to communicate directly with the Borland Database Engine. To access or set the SQL statement that this query component executes, use the *SQL* property. To access or set the parameters used in a parameterized SQL statement, use the *Params* property.

See also *Params* property, *SQL* property

StmtHandle

Identifies the Borland Database Engine (BDE) statement handle for the query.

```
property StmtHandle: HDBIStmt;
```

Description Retrieve *StmtHandle* if an application makes a direct call to the BDE, bypassing the methods of *TQuery*. Some BDE API calls require a statement handle as a parameter. Under all other circumstances an application does not need to access this property.

Text

Points to the actual text of the SQL query passed to the Borland Database Engine (BDE).

```
property Text: PChar;
```

Description *Text* is a read-only property that can be examined to determine the actual contents of SQL statement passed to the BDE. For parameterized queries, *Text* contains the SQL statement with parameters replaced by the parameter substitution symbol (?)

In general there should be no need to examine the *Text* property. To access or change the SQL statement for the query, use the *SQL* property. To examine or modify parameters, use the *Params* property.

See also *Params* property, *SQL* property

UniDirectional

Determines whether or not Borland Database Engine (BDE) bidirectional cursors are enabled for a query's result set.

```
property UniDirectional: Boolean;
```

Description Set *UniDirectional* to control whether or not a BDE cursor can move forward and backward through a result set. By default *UniDirectional* is *False*, enabling forward and backward navigation.

Note Traditionally SQL cursors are unidirectional. They can travel only forward through a dataset. The BDE, however, permits bidirectional travel by caching records. If an application does not need bidirectional access to records in a result set, set *UniDirectional* to *True*. When *UniDirectional* is *True*, an application requires less memory and performance is improved.

Events

The following table lists the events of *TQuery*:

Event	Visibility	Documented in
AfterCancel	published	TDataSet
AfterClose	published	TDataSet
AfterDelete	published	TDataSet
AfterEdit	published	TDataSet
AfterInsert	published	TDataSet
AfterOpen	published	TDataSet
AfterPost	published	TDataSet
AfterScroll	published	TDataSet
BeforeCancel	published	TDataSet
BeforeClose	published	TDataSet
BeforeDelete	published	TDataSet
BeforeEdit	published	TDataSet
BeforeInsert	published	TDataSet
BeforeOpen	published	TDataSet
BeforePost	published	TDataSet

Event	Visibility	Documented in
BeforeScroll	published	TDataSet
OnCalcFields	published	TDataSet
OnDeleteError	published	TDataSet
OnEditError	published	TDataSet
OnFilterRecord	published	TDataSet
OnNewRecord	published	TDataSet
OnPostError	published	TDataSet
OnServerYield	published	TBDEDataSet
OnUpdateError	published	TBDEDataSet
OnUpdateRecord	published	TBDEDataSet

Methods

The following table lists the methods of *TQuery*:

Method	Visibility	Documented in
ActiveBuffer	public	TDataSet
Append	public	TDataSet
AppendRecord	public	TDataSet
ApplyUpdates	public	TBDEDataSet
Assign	public	TPersistent
BookmarkValid	public	TBDEDataSet
Cancel	public	TBDEDataSet
CancelUpdates	public	TBDEDataSet
CheckBrowseMode	public	TDataSet
CheckOpen	public	TDBDataSet
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClearFields	public	TDataSet
Close	public	TDataSet
CommitUpdates	public	TBDEDataSet
CompareBookmarks	public	TBDEDataSet
ConstraintCallBack	public	TBDEDataSet
ControlsDisabled	public	TDataSet
Create	public	TQuery
CursorPosChanged	public	TDataSet
DefaultHandler	public	TObject

Method	Visibility	Documented in
Delete	public	TDataSet
Destroy	public	TQuery
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableConstraints	public	TBDEDataSet
DisableControls	public	TDataSet
Dispatch	public	TObject
Edit	public	TDataSet
EnableConstraints	public	TBDEDataSet
EnableControls	public	TDataSet
ExecSQL	public	TQuery
FetchAll	public	TBDEDataSet
FieldAddress	public	TObject
FieldByName	public	TDataSet
FindComponent	public	TComponent
FindField	public	TDataSet
FindFirst	public	TDataSet
FindLast	public	TDataSet
FindNext	public	TDataSet
FindPrior	public	TDataSet
First	public	TDataSet
Free	public	TObject
FreeBookmark	public	TDataSet
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetBookmark	public	TDataSet
GetCurrentRecord	public	TBDEDataSet
GetFieldList	public	TDataSet
GetFieldNames	public	TDataSet
GetIndexInfo	public	TBDEDataSet
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GotoBookmark	public	TDataSet
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
Insert	public	TDataSet
InsertComponent	public	TComponent

Method	Visibility	Documented in
InsertRecord	public	TDataSet
InstanceSize	public	TObject
IsEmpty	public	TDataSet
IsLinkedTo	public	TDataSet
Last	public	TDataSet
Locate	public	TBDEDataSet
Lookup	public	TBDEDataSet
MethodAddress	public	TObject
MethodName	public	TObject
MoveBy	public	TDataSet
NewInstance	public	TObject
Next	public	TDataSet
Open	public	TDataSet
ParamByName	public	TQuery
Post	public	TDataSet
Prepare	public	TQuery
Prior	public	TDataSet
Refresh	public	TDataSet
RemoveComponent	public	TComponent
Resync	public	TDataSet
RevertRecord	public	TBDEDataSet
SafeCallException	public	TComponent
SetFields	public	TDataSet
Translate	public	TDataSet
UnPrepare	public	TQuery
UpdateCursorPos	public	TDataSet
UpdateRecord	public	TDataSet
UpdateStatus	public	TBDEDataSet

Create

Creates an instance of a query component.

constructor Create(AOwner: TComponent);

Description Call *Create* to instantiate a query declared in an application. *Create* calls its inherited *Create* constructor, creates an empty SQL statement list, an empty parameter list, sets the *OnChange* event handler for the SQL statement list, establishes a data link, sets the *RequestLive* property to *False*, sets the *ParamCheck* property to *True*, and sets the *RowsAffected* property to –1.

See also *Destroy* method, *ParamCheck* property, *Params* property, *RequestLive* property, *RowsAffected* property, *TStringList.OnChange* event

Destroy

Destroys the instance of a query.

destructor Destroy;

Description Do not call *Destroy* directly. Instead call *Free* to verify that the query is not already freed before calling *Destroy*. *Destroy* disconnects from the server, frees the SQL statement list, the parameter list, and the data link and SQL binary storage area, and then calls its inherited *Destroy* destructor.

See also *Create* method, *Params* property, *SQL* property, *SQLBinary* property

ExecSQL

Executes the SQL statement for the query.

procedure ExecSQL;

Description Call *ExecSQL* to execute the INSERT, UPDATE, or DELETE statement currently assigned to the *SQL* property. *ExecSQL* is also used to execute data definition statements.

Note For SELECT statements, call *Open* instead of *ExecSQL*.

ExecSQL prepares the statement in *SQL* property for execution if it has not already been prepared. To speed performance, an application should ordinarily call *Prepare* before calling *ExecSQL* for the first time.

See also *Open* method, *Prepare* method, *SQL* property

ParamByName

Q

Accesses parameter information based on a specified parameter name.

function ParamByName(**const** Value: **string**): TParam;

Description Call *ParamByName* to set or use parameter information for a specific parameter based on its name. *Value* is the name of the parameter for which to retrieve information. *ParamByName* is used to set an parameter's value at runtime. For example, the following statement retrieves the current value of a parameter called "Contact" into an edit box:

```
Edit1.Text := Query1.ParamsByName('Contact').AsString;
```

See also *Params* property

Prepare

Sends a query to the Borland Database Engine (BDE) and the server for optimization prior to execution.

procedure Prepare;

Description Call *Prepare* to have the BDE and a remote database server allocate resources for the query and to perform additional optimizations. Calling *Prepare* before executing a query improves application performance.

Note Delphi automatically prepares a query if it is executed without first being prepared. After execution, Delphi unprepares the query. When a query will be executed a number of times, an application should always explicitly prepare the query to avoid multiple and unnecessary prepares and unprepares.

Preparing a query consumes some database resources, so it is good practice for an application to unprepare a query once it is done using it. The *UnPrepare* method unprepares a query.

Note When you change the text of a query at runtime, the query is automatically closed and unprepared.

See also *ExecSQL* method, *Open* method, *UnPrepare* method

UnPrepare

Frees the resources allocated for a previously prepared query.

```
procedure UnPrepare;
```

Description Call *UnPrepare* to free the resources allocated for a previously prepared query on the server and client sides.

Preparing a query consumes some database resources, so it is good practice for an application to unprepare a query once it is done using it. The *UnPrepare* method unprepares a query.

Note When you change the text of a query at runtime, the query is automatically closed and unprepared.

See also *ExecSQL* method, *Open* method, *Prepare* method

TQueryTableProducer

Unit: DBWeb | Palette: Internet

TObject • TPersistent • TComponent • TCustomContentProducer • TDSTableProducer • TQueryTableProducer

TQueryTableProducer assembles a sequence of HTML commands to display the records from a *TQuery* object in a tabular format.

Description Use *TQueryTableProducer* to produce an HTML table for the records of a parameterized query. The *TQuery* object obtains its parameters from an HTTP request message. If the request is a GET message, the parameters for the query are the *QueryFields* property of the request object. If the request is a POST message, the parameters for the query are the *ContentFields* property of the request object.

See also *TDataSetTableProducer* object, *TQuery* object, *TWebRequest.ContentFields* property, *TWebRequest.MethodType* property, *TWebRequest.QueryFields* property

Properties

The following table lists the properties of *TQueryTableProducer*:

Property	Visibility	Default	Documented in
Caption	published		TDSTableProducer
CaptionAlignment	published	caDefault	TDSTableProducer
Columns	published		TDSTableProducer
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DataSet	public		TQueryTableProducer
DesignInfo	public		TComponent
Dispatcher	published		TCustomContentProducer
Editor	public		TDSTableProducer
Footer	published		TDSTableProducer
Header	published		TDSTableProducer
MaxRows	published	20	TDSTableProducer
Name	published		TComponent
Owner	public, read-only		TComponent
Query	published		TQueryTableProducer
RowAttributes	published		TDSTableProducer
TableAttributes	published		TDSTableProducer
Tag	published	0	TComponent
VCLComObject	public		TComponent

DataSet

DataSet is the *TQuery* object that *TQueryTableProducer* uses to obtain its data.

property DataSet: TDataSet;

Description Set *DataSet* to specify the query to run to obtain the data for the HTML table. Use *DataSet* to set the query for the *TQueryTableProducer* rather than the *DataSource* property. *TQueryTableProducer* does not use an external *TDataSource* object because it does not maintain record currency with other data objects in the application.

The individual field bindings specified by the *THTMLTableColumn* objects in the *Columns* property refer to fields in *DataSet*.

Note *DataSet* is identical to the *Query* property, except the query object is viewed as a *TDataSet* type.

See also *Columns* property, *DataSource* property, *Query* property, *THTMLTableColumn.FieldName* property

Query

Query is the *TQuery* object that *TQueryTableProducer* runs to obtain its data.

```
property Query: TQuery;
```

Description Set *Query* to specify the query to run to obtain the data for the HTML table. Use *Query* to set the query for the *TQueryTableProducer* rather than the *DataSource* property. *TQueryTableProducer* does not use an external *TDataSource* object because it does not maintain record currency with other data objects in the application.

The individual field bindings specified by the *THTMLTableColumn* objects in the *Columns* property refer to fields in *DataSet*.

Note *Query* is identical to the *DataSet* property, except *Query* does not use the *TDataSet* base class of the query object.

See also *Columns* property, *DataSet* property, *DataSource* property, *THTMLTableColumn.FieldName* property

Events

The following table lists the events of *TQueryTableProducer*:

Event	Visibility	Documented in
OnCreateContent	published	TDSTableProducer
OnFormatCell	published	TDSTableProducer
OnGetTableCaption	published	TDSTableProducer

Methods

The following table lists the methods of *TQueryTableProducer*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginUpdate	public	TDSTableProducer
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject

Method	Visibility	Documented in
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Content	public	TQueryTableProducer
ContentFromStream	public	TCustomContentProducer
ContentFromString	public	TCustomContentProducer
Create	public	TDSTableProducer
DefaultHandler	public	TObject
Destroy	public	TDSTableProducer
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
EndUpdate	public	TDSTableProducer
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Content

Content returns the HTML representation of the query results.

function Content: string; **override**;

Description Call *Content* to run the query with the parameters taken from the HTTP request message and retrieve the resulting HTML representation. If the

MethodType of the request is *mtGet*, the parameters for the query are the *QueryFields* property of the request. If the *MethodType* of the request is *mtPost*, the parameters for the query are the *ContentFields* property of the request.

After running the query, *Content* calls the *OnCreateContent* event handler. If the *OnCreateContent* event handler indicates that the *Content* method should continue, *Content* returns the *Header* property, followed by an HTML table image of the records in the query, followed by the *Footer* property. If the *OnCreateContent* event handler indicates that the *Content* method should not continue, *Content* returns an empty string.

The string returned by *Content* can be used as the *Content* property of the response object associated with the request message.

See also *Footer* property, *Header* property, *OnCreateContent* event, *TWebRequest.ContentFields* property, *TWebRequest.MethodType* property, *TWebRequest.QueryFields* property, *TWebResponse.Content* property

TRadioButton

Unit: StdCtrls | Palette: Standard

TObject • TPersistent • TComponent • TControl • TWinControl • TButtonControl • TRadioButton

TRadioButton is a wrapper for a Windows radio button.

Description Use *TRadioButton* to add a radio button to a form. Use radio buttons to present a set of mutually exclusive options to the user—that is, only one radio button in a set can be selected at any time. When the user selects a radio button, the previously selected radio button becomes unselected. Radio buttons are frequently grouped in a radio group box (*TRadioGroup*). Add the group box to the form first, then choose the radio buttons from the Component palette and put them in the group box.

By default, all radio buttons that are directly contained by the same windowed control container, such as a *TRadioGroup* or *TPanel*, are grouped. For example, two radio buttons on a form can be checked at the time only if they are contained in separate containers, such as two different group boxes.

See also *TCheckBox* object, *TPanel* object, *TRadioGroup* object

Properties

The following table lists the properties of *TRadioButton*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
Alignment	published	taRightJustify	TRadioButton

Property	Visibility	Default	Documented in
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Caption	published		TControl
Checked	published	False	TRadioButton
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	True	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ShowHint	published		TControl
Showing	public, read-only		TWinControl

Property	Visibility	Default	Documented in
TabOrder	published	–1	TWinControl
TabStop	published	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Alignment

Alignment determines the position of the caption relative to the radio button.

```
type
  TAlignment = (taLeftJustify, taRightJustify, taCenter);
  TLeftRight = taLeftJustify..taRightJustify;

property Alignment: TLeftRight;
```

Description Set *Alignment* to determine where the caption of the radio button should appear. These are the possible values:

Value	Meaning
taLeftJustify	The caption appears to the left of the radio button.
taRightJustify	The caption appears to the right of the radio button.

Note Do not confuse the *Alignment* property with the alignment of the caption in the text area of the radio button. The caption always appears left-aligned. *Alignment* only controls on which side of the button the text area appears.

See also *Caption* property

Checked

Checked determines whether the option represented by the radio button is selected.

```
property Checked: Boolean;
```

Description Read *Checked* to determine whether the radio button is selected. Set *Checked* to *True* to select the radio button and deselect all other radio buttons in the same container. Set *Checked* to *False* to deselect the radio button, leaving no radio button in the group selected.

See also *TRadioGroup* object

Events

The following table lists the events of *TRadioButton*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TRadioButton*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TRadioButton
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TWinControl

Method	Visibility	Documented in
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl

Method	Visibility	Documented in
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

Create

Create creates an instance of *TRadioButton*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to instantiate a radio button at runtime. Radio buttons added to forms at design time are created automatically. *Create* calls the inherited constructor, then sets the initial values for the radio button, such as the *Width* and *Height* property values to 113 pixels and 17 pixels, respectively, the *ControlStyle* set to *csSetCaption* and *csDoubleClicks*, and the *Alignment* property value to *taLeftJustify*.

See also *Alignment* property, *ControlStyle* property, *Height* property, *Width* property

TRadioGroup

R

Unit: ExtCtrls Palette: Standard

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomControl • TCustomGroupBox • TCustomRadioGroup • TRadioGroup

TRadioGroup represents a group of radio buttons that function together.

Description A *TRadioGroup* object is a special group box that contains only radio buttons. Radio buttons that are placed directly in the same control component are said to be "grouped." When the user checks a radio button, all other radio buttons in its group become unchecked. Hence, two radio buttons on a form can be checked at the same time only if they are placed in separate containers, such as group boxes.

To add radio buttons to a *TRadioGroup*, edit the *Items* property in the Object Inspector. Each string in *Items* makes a radio button appear in the group box

with the string as its caption. The value of the *ItemIndex* property determines which radio button is currently selected.

You can display the radio buttons in a single column or in multiple columns by setting the value of the *Columns* property.

See also *TCustomRadioGroup* object, *Items* property, *ItemIndex* property, *Columns* property, *TDBRadioGroup* object

Properties

The following table lists the properties of *TRadioGroup*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Caption	published		TControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
Columns	published	1	TCustomRadioGroup
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl

Property	Visibility	Default	Documented in
ItemIndex	published	–1	TCustomRadioGroup
Items	published		TCustomRadioGroup
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	True	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Events

The following table lists the events of *TRadioGroup*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TRadioGroup*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomRadioGroup
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TCustomRadioGroup
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent

Method	Visibility	Documented in
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

TRemoteServer

Unit: DBClient

Palette: Data Access

TObject • TPersistent • TComponent • TCustomRemoteServer • TRemoteServer

TRemoteServer is the object for handling a remote server connection in a client application that is part of a multi-tiered application.

Description

TRemoteServer is used in the client portion of a multi-tiered database application to establish and maintain the connection between the client and a remote application server. The remote server object can

- Establish an initial connection to a remote application server.
- Provide a list of provider names available to client datasets through this connection.
- Obtain an *IProvider* interface based on a provider name.
- Maintain a list of local client datasets that can be serviced through this connection.
- Drop the connection to the remote application server.

The remote server object establishes the initial connection between the client application and a remote application server using DCOM. Once the connection is established, the client registers any or all of its client datasets with the remote server, and these client datasets can then communicate directly with the application server's *TProvider* object through its *IProvider* DCOM interface.

See also

TProvider object, *TClientDataSet* object

Properties

The following table lists the properties of *TRemoteServer*:

Property	Visibility	Default	Documented in
AppServer	public, read-only		TRemoteServer
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ComputerName	published		TRemoteServer
Connected	published	False	TRemoteServer
DesignInfo	public		TComponent
Name	published		TComponent

Property	Visibility	Default	Documented in
Owner	public, read-only		TComponent
ServerName	published		TRemoteServer
Tag	published	0	TComponent
VCLComObject	public		TComponent

AppServer

Indicates the variant containing an interface returned to the remote server.

property AppServer: Variant;

Description *ServerDispatch* is a read-only property that points to the interface returned to the remote server by an application server's provider component. Client applications can communicate directly with the application server through this interface.

See also *TClientDataSet* object

ComputerName

Specifies the name of the machine on which a multi-tiered application's application server is located.

property ComputerName: **string**;

Description Set *ComputerName* to the machine name of the computer where the application server resides. If *ComputerName* is blank (the default), Delphi assumes the application server can be found on the user's machine or that the computer name can be derived from the system registry based on DCOM or OLE Enterprise configuration. Normally the client and the application server run on different machines, so *ComputerName* must be set by one of these methods.

See also *ServerName* property

Connected

Connected indicates whether the remote server object is connected to a remote application server.

property Connected: Boolean;

Description Use *Connected* to read or set the current connection status for a remote server object. By default, *Connected* is *False*.

See also *GetConnected* method, *SetConnected* method

ServerName

Specifies the name of the application server to which the remote server connects.

```
property ServerName: string;
```

Description Use *ServerName* to specify the name of the application server to which the remote server object should connect. At design time, the developer can choose an available server from the drop-down list for this property in the Object Inspector.

If *ServerName* is blank (the default), Delphi assumes that server name can be derived from the system registry based on DCOM or OLE Enterprise configuration.

See also *ComputerName* property

Events

The following table lists the events of *TRemoteServer*:

Event	Visibility	Documented in
AfterConnect	published	TCustomRemoteServer
AfterDisconnect	published	TCustomRemoteServer
BeforeConnect	published	TCustomRemoteServer
BeforeDisconnect	published	TCustomRemoteServer

Methods

The following table lists the methods of *TRemoteServer*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TCustomRemoteServer
DefaultHandler	public	TObject
Destroy	public	TCustomRemoteServer
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject

Method	Visibility	Documented in
DoConnect	protected	TRemoteServer
DoDisconnect	protected	TRemoteServer
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetConnected	protected	TRemoteServer
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetProvider	public	TRemoteServer
GetProviderNames	public	TRemoteServer
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

DoConnect

DoConnect establishes a connection to an application server.

procedure DoConnect;

Description *DoConnect* is called internally by *SetConnected*. *DoConnect* establishes an ActiveX dual interface between the remote server and a provider object on the application server.

See also *AfterConnect* event, *BeforeConnect* event, *SetConnected* method

DoDisconnect

DoDisconnect drops a connection to an application server.

procedure DoDisonnect;

Description *DoDisconnect* is called internally by *SetConnected*. *DoDisconnect* drops the connection to an application server.

See also *AfterDisconnect* event, *BeforeDisconnect* event, *SetConnected* method

GetConnected

GetConnected is the read method for the *Connected* property.

```
function GetConnected: Boolean;
```

Description *GetConnected* determines if the remote server is connected to an application server. The value returned by *GetConnected* becomes the current setting of the *Connected* property.

If *GetConnected* returns *True*, a connection is established. Otherwise, *GetConnected* returns *False*.

See also *Connected* property, *SetConnected* method

GetProvider

GetProvider is a function that returns an *IProvider* interface implemented by the provider object on the application server in response to a client application connection request.

```
function GetProvider (const ProviderName: string): IProvider;
```

Description Call *GetProvider* to return an *IProvider* corresponding to the *ProviderName* parameter. The *IProvider* interface returned by *GetProvider* is subsequently used by client datasets to communicate directly with the provider object for data exchange.

ProviderName is the name of the property on the application server which returns the *IProvider* interface.

For more information about using COM interfaces in Delphi, see the *Developer's Guide*.

See also *AddDataSet* method, *TClientDataSet* object, *TProvider* object

GetProviderNames

GetProviderNames retrieves a list of all available provider objects from an application server into a list created and maintained by the remote server.

```
procedure GetProviderNames(List: TStrings);
```

Description Call *GetProviderNames* to return a list of all available *IProvider* interfaces established by the application server. Interfaces can then be made available to client datasets that use this remote server.

List is a string list, created and maintained by the client application, into which a list of *IProvider* interfaces is stored.

See also *GetProvider* method, *TClientDataset* object, *TProvider* object

TReplaceDialog

Unit: Dialogs | Palette: Dialogs

TObject • TPersistent • TComponent • TCommonDialog • TFindDialog • TReplaceDialog

TReplaceDialog displays a search-and-replace dialog.

Description *TReplaceDialog* is a special version of *TFindDialog* that prompts the user for both a search string and a replace string. Like the Find dialog, the Replace dialog is modeless and does not appear at runtime until activated by a call to the *Execute* method.

See Also *TFindDialog* object

Properties

The following table lists the properties of *TReplaceDialog*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Ctl3D	published	True	TCommonDialog
DesignInfo	public		TComponent
FindText	published		TFindDialog
Handle	public, read-only		TCommonDialog
HelpContext	published	0	TCommonDialog
Left	public		TFindDialog
Name	published		TComponent
Options	published	[frDown]	TFindDialog
Owner	public, read-only		TComponent
Position	public		TFindDialog
ReplaceText	published		TReplaceDialog
Tag	published	0	TComponent
Top	public		TFindDialog
VCLComObject	public		TComponent

R

ReplaceText

Contains the text that should replace *FindText*.

property ReplaceText: **string**;

FindText, taken from the Find What edit box, contains the text string that the user wants to search for. *ReplaceText*, taken from the Replace With edit box, contains the text string that will replace *FindText*. To make a default text string appear in the Replace With edit box when the dialog opens, assign a value to *ReplaceText* in the Object Inspector or in program code.

See also *FindText* property

Events

The following table lists the events of *TReplaceDialog*:

Event	Visibility	Documented in
OnClose	published	TCommonDialog
OnFind	published	TFindDialog
OnReplace	published	TReplaceDialog
OnShow	published	TCommonDialog

OnReplace

OnReplace occurs when the user clicks the Replace or Replace All button.

type TNotifyEvent = **procedure**(Sender: TObject) **of object;**

property OnReplace: TNotifyEvent;

Description *OnReplace* occurs whenever the user clicks Replace or Replace All. In addition, when the user clicks Replace, the *frReplace* flag is set (and the *frReplaceAll* flag is turned off) in *Options*. When the user clicks ReplaceAll, the *frReplaceAll* flag is set (and the *frReplace* flag is turned off) in *Options*.

Write an *OnReplace* event handler that searches for the text specified in *FindText* and replaces it with the text in *ReplaceText*. Use the *Options* flags to determine how the search is conducted.

For example, the following event handler searches a *TMemo* object called *Memo1* and replaces *FindText* with *ReplaceText*. It uses *TMemo*'s *SelStart*, *SelLength*, and *SelText* properties.

```
procedure TForm1.ReplaceDialog1Replace(Sender: TObject);
var
  SelPos: Integer;
begin
  with TReplaceDialog(Sender) do
```

```
  begin
    { Perform a global case-sensitive search for FindText in Memo1 }
    SelPos := Pos(FindText, Memo1.Lines.Text);
    if SelPos > 0 then
    begin
      Memo1.SelStart := SelPos - 1;
      Memo1.SelLength := Length(FindText);
      { Replace selected text with ReplaceText }
      Memo1.SelText := ReplaceText;
    end
    else MessageDlg(Concat('Could not find "', FindText, '" in Memo1.'), mtError,
      [mbOk], 0);
  end;
end;
```

See Also *OnFind* event, *Options* property, *ReplaceText* property, *FindText* property, *TMemo* object, *TNotifyEvent* type

Methods

The following table lists the methods of *TReplaceDialog*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
CloseDialog	public	TFindDialog
Create	public	TReplaceDialog
DefaultHandler	public	TObject
Destroy	public	TFindDialog
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
Execute	public	TFindDialog
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject

Method	Visibility	Documented in
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Create

Creates and initializes a *TReplaceDialog* instance.

constructor Create(AOwner: TComponent); **override**;

Description The *Create* method generates a *TReplaceDialog* instance, but the new dialog does not appear on the form at runtime until the *Execute* method is called.

See also *Execute* method

TReport

Unit: Report

TObject • TPersistent • TComponent • TReport

TReport is a wrapper for Borland's ReportSmith application.

Description Use a *TReport* object to enable an application to launch ReportSmith for designing and running reports.

Double-click a *TReport* object at design time to begin running ReportSmith. To bring up particular report, specify the existing report name using the Object Inspector and choose Cancel when the Open Report dialog box appears. ReportSmith minimizes. Click the ReportSmith icon to restore it, and the specified report displays.

To run a specified report at runtime, call the *Run* method.

For information about using ReportSmith, see the ReportSmith documentation.

Properties

The following table lists the properties of *TReport*:

Property	Visibility	Default	Documented in
AutoUnload	published	False	TReport
Busy	public, read-only		TReport
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DesignInfo	public		TComponent
EndPage	published	9999	TReport
InitialValues	published		TReport
LaunchType	published	ltDefault	TReport
MaxRecords	published	0	TReport
Name	published		TComponent
Owner	public, read-only		TComponent
Preview	published	False	TReport
PrintCopies	published	1	TReport
ReportDir	published		TReport
ReportHandle	public, read-only		TReport
ReportName	published		TReport
StartPage	published	1	TReport
Tag	published	0	TComponent
VCLComObject	public		TComponent
VersionMajor	public, read-only		TReport
VersionMinor	public, read-only		TReport

R

AutoUnload

AutoUnload specifies whether ReportSmith unloads from memory when the specific report is finished running.

property AutoUnload: Boolean;

Description

Set *AutoUnload* to *True*, to unload ReportSmith as soon as the report is finished running. Set *AutoUnload* to *False* to keep the ReportSmith application running after the report is finished.

For example, consider an application that includes a menu item to run a report. After the report runs, ReportSmith should stay in memory so the report can be run again quickly. To remove ReportSmith from memory when *AutoUnload* is *False*, call the *CloseApplication* method.

The following code sets *AutoUnload* to *False*, so that *Report1* can be run twice using two different variables. After the second run, ReportSmith is unloaded by a call to *CloseApplication*.

```
Report1.AutoUnload := False;
if Report1.SetVariable('FName', 'Linda') then
  Report1.Run;
if Report1.SetVariable('LName', 'King') then
  Report1.Run;
Report1.CloseApplication(False);
```

See also *CloseApplication* method, *Run* method

Busy

Busy indicates if ReportSmith is read to perform an operation.

property Busy: Boolean;

Description Read *Busy* to determine whether ReportSmith is ready to execute a report command. When *Busy* is *True*, none of the report command methods will succeed.

See also *CloseApplication* method, *CloseReport* method, *Connect* method, *Print* method, *RecalcReport* method, *Run* method, *RunMacro* method, *SetVariable* method, *SetVariableLines* method

EndPage

EndPage specifies the last page of the report to be printed.

property EndPage: Word;

Description Set *EndPage* to stop printing before the entire report has printed. If the report is fewer than 9999 pages and *EndPage* is not set, the entire report is printed. Limiting the number of pages is especially useful when designing a report, so that formatting can be checked without tying up a printer.

See also *Preview* property, *Print* method, *StartPage* property

InitialValues

InitialValues is a list of report variable strings the report uses to run.

property InitialValues: TStrings;

Description Set *InitialValues* to initialize report variables that are used by the ReportSmith report. Each report variable is set by a string with the format

```
@VariableName=<Value>
```

where *VariableName* is the name of the report variable and *Value* is the value to initialize it to.

The *InitialValues* are passed to ReportSmith when the *Run* method is called. To use *InitialValues* when printing a report, set *Preview* to *False* and call the *Run* method, rather than calling the *Print* method.

When a report uses report values and *InitialValues* is not set, ReportSmith brings up dialog boxes to prompt the user for these values.

Note See the ReportSmith documentation for the use of report variables.

See also *Preview* property, *Print* method, *Run* method, *SetVariable* method, *SetVariableLines* method

LaunchType

LaunchType specifies whether the ReportSmith application will be launched in design mode or run mode.

```
type TLaunchType = (ltDefault, ltRunTime, ltDesignTime);
```

```
property LaunchType: TLaunchType;
```

Description Set *LaunchType* to specify whether the user will be able to modify the report and create new reports, or whether Reportsmith will only print or preview reports.

The values for *LaunchType* and their effects are given in the following table.

Value	Meaning
ltDefault	At runtime, launch ReportSmith Runtime for printing and previewing reports. At design time, Launch ReportSmith to modify or create a report.
ltRunTime	Launch ReportSmith Runtime for printing and previewing reports.
ltDesignTime	Launch ReportSmith to modify or create a report.

See also *Run* method

R

MaxRecords

MaxRecords specifies the number of database records to use in a report.

```
property MaxRecords: Word;
```

Description Set *MaxRecords* to limit the number of records that will be used for the report. For example, to see a sample report on a database containing 50,000 records, specify a *MaxRecords* value that limits the number of records in the report to a much smaller number. Setting *MaxRecords* to 0 tells ReportSmith to include all records in the report.

Note *MaxRecords* will affect the size of the report. The *EndPage* property will not limit the number of pages from the report that are printed, but will not change the report length. Thus, unlike *EndPage*, setting *MaxRecords* will still allow the report footer to print.

See also *EndPage* property

Preview

Preview specifies whether a report is viewed onscreen or printed.

property Preview: Boolean;

Description Set *Preview* to *True* to display the report on screen when the *Run* method is called. Set *Preview* to *False* to print the report when the *Run* method is called.

If *Preview* is set to *False* at design time, the ReportSmith application is minimized when it is launched at runtime. Even when *Preview* is *True*, the report can still be printed by calling the *Print* method.

See also *Print* method, *Run* method

PrintCopies

PrintCopies is the number of copies of the report that are printed.

property PrintCopies: Word;

Description Set *PrintCopies* to print out multiple copies of a report.

See also *Preview* property, *Print* method

ReportDir

ReportDir is the directory where ReportSmith stores its reports and looks for saved reports.

property ReportDir: string;

Description Specify a report directory to indicate where an application stores its reports. *ReportDir* is especially useful when using the *TReport* object to print or preview multiple report. When *ReportDir* is set, the *ReportName* does not need to include the path.

See also *ReportName* property

ReportHandle

ReportHandle is the window handle for the ReportSmith application.

property ReportHandle: HWND;

Description Read *ReportHandle* to get a window handle for Windows API function calls that involve the application window of the ReportSmith application. For example, the following code retrieves the window placement information for ReportSmith, given Report1 is a valid *TReport* object.

```
var
  RSWinPlacement: PWindowPlacement;
begin
  GetWindowPlacement(Report1.ReportHandle, RSWinPlacement);
end;
```

ReportName

ReportName is the name of the report to run when the *Run* method is called.

property ReportName: **string**;

Description Use *ReportName* to preview or print a particular report. If the report is in the *ReportDir*, only the file name is required. Otherwise, specify the full path name of the report. If the *ReportName* does not include a file extension, .RPT is assumed.

The following code runs a report specified by the user in the Open dialog box.

```
if OpenDialog1.Execute then
begin
  Report1.ReportName := OpenDialog1.FileName;
  Report1.Run;
end;
```

See also *ReportDir* property

StartPage

StartPage specifies the first page of the report to be printed.

property StartPage: Word;

Description Set *StartPage* to skip some of the first pages of the report when printing.

See also *EndPage* property, *Preview* property, *Print* method

R

VersionMajor

VersionMajor identifies which major version of ReportSmith is running.

property VersionMajor: Integer;

Description Read *VersionMajor* to determine the major version number of the ReportSmith application. For example, if ReportSmith 3.0 is running, the value of *VersionMajor* is 3. The minor version value is reported in the *VersionMinor* property.

See also *VersionMinor* property

VersionMinor

VersionMinor identifies which minor version of ReportSmith is running.

property VersionMinor: Integer;

Read *VersionMinor* to determine the minor version number of the ReportSmith application. For example, if ReportSmith 3.0 is running, the value of *VersionMinor* is 0. The major version value is reported in the *VersionMajor* property.

See also *VersionMajor* property

Methods

The following table lists the methods of *TReport*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
CloseApplication	public	TReport
CloseReport	public	TReport
Connect	public	TReport
Create	public	TReport
DefaultHandler	public	TObject
Destroy	public	TReport
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent

Method	Visibility	Documented in
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Print	public	TReport
RecalcReport	public	TReport
RemoveComponent	public	TComponent
Run	public	TReport
RunMacro	public	TReport
SafeCallException	public	TComponent
SetVariable	public	TReport
SetVariableLines	public	TReport

CloseApplication

CloseApplication shuts down ReportSmith if it is running.

```
function CloseApplication(ShowDialogs: Boolean): Boolean;
```

Description Use *CloseApplication* to manually shut down ReportSmith when *AutoUnload* is *False*. *CloseApplication* attempts to terminate ReportSmith and then looks for confirmation from ReportSmith in return. If *CloseApplication* returns *True*, ReportSmith terminated successfully; if it returns *False*, ReportSmith was not able to close at the current time.

The value of the *ShowDialogs* parameter determines whether ReportSmith displays dialog boxes prompting users to save the existing report before closing, and so on.

To close a report without shutting down ReportSmith, use the *CloseReport* method.

Note *CloseApplication* will fail if the *Busy* property is *True*.

See also *AutoUnload* property, *Busy* property, *CloseReport* method

CloseReport

CloseReport tells ReportSmith to terminate the report.

```
function CloseReport(ShowDialogs: Boolean): Boolean;
```

Description Call *CloseReport* to close a report that is currently running. *CloseReport* tells ReportSmith to terminate the report and then waits for confirmation in

return. If *CloseReport* returns *True*, ReportSmith received the message and closed the report. If *CloseReport* returns *False*, ReportSmith could not terminate the report at the current time.

The *ShowDialogs* parameter determines whether dialog boxes that prompt the user about saving the report appear before the report closes, and so on.

The following code terminates the running report if the user chooses Yes from a dialog box:

```
if MessageDlg('Do you want to stop running ' + Report1.ReportName + ' ?',
  mtConfirmation, [mbYes, mbNo], 0) = mrYes then
  if Report1.CloseReport(False) then MessageDlg(Report1.ReportName + ' canceled.',
    mtInformation, [mbOK], 0);
```

Note *CloseReport* will fail if the *Busy* property is *True*.

See also *Busy* property, *Run* method

Connect

Connect connects to a database server, bypassing the ReportSmith login dialog.

```
function Connect(ServerType: Word; const ServerName, UserName, Password,
  DatabaseName: string): Integer;
```

Description Use *Connect* to log in to the database server before running a report that uses tables on a server that requires a password. If *Connect* is not called before running such a report, ReportSmith will bring up a dialog prompting the user to log in to the server.

The *ServerType* parameter is the type of server, such as INTERBASE or ORACLE. The *ServerName* parameter is the name of the specific server. The *UserName* parameter is the user's account name on the server. The *Password* parameter is the log-in password, and the *DatabaseName* parameter is the name of the database.

Note *Connect* will fail if the *Busy* property is *True*.

See also *Busy* property

Create

Create creates an instance of *TReport*.

```
constructor Create(AOwner: TComponent); override;
```

Description Call *Create* to instantiate a *TReport* object at runtime. For *TReport* objects placed on forms or data modules at design time, *Create* is called automatically.

Create allocates memory for a *TReport* object and initializes the following properties.

- PrintCopies is set to 1.
- StartPage is set to 1.
- EndPage is set to 9999.
- MaxRecords is set to 0.
- LaunchType is set to ltDefault.

See also *EndPage* property, *LaunchType* property, *MaxRecords* property, *PrintCopies* property, *StartPage* property

Destroy

Destroy destroys an instance of *TReport*.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the *TReport* object is not already freed, and only then calls *Destroy*. Do not call *Destroy* when the *Busy* property is *True*.

Destroy calls *CloseApplication* to shut down ReportSmith before freeing the memory for the *TReport* object.

See also *Busy* property, *CloseApplication* method, *Free* method

Print

Print tells ReportSmith to print the current report.

function Print: Boolean;

Description Call *Print* to print the report after it has been loaded into *ReportSmith*. If *Print* returns *True*, ReportSmith received the message to print the report. If *Print* returns *False*, ReportSmith could not receive the message at the current time.

Print tells ReportSmith to print the current report. This may not be the same as the report specified by *ReportName* if the user has loaded a different report from within ReportSmith. To ensure that the report specified by *ReportName* is printed, use the *Run* method with *Preview* set to *False*.

Note *Print* will fail if the *Busy* property is *True*.

See also *Busy* property, *Preview* property, *ReportName* property, *Run* method

RecalcReport

RecalcReport recalculates and reruns the report.

function RecalcReport: Integer;

Description Call *RecalcReport* to tell ReportSmith to recalculate and reprint the report with a new value for a report variable previously changed with the *SetVariable* method.

RecalcReport sends a message to ReportSmith and looks for confirmation in return. If *RecalcReport* returns *0*, the message to recalculate the report was sent successfully to ReportSmith. If it returns an error code, ReportSmith could not recalculate the report at the current time.

For more information about report variables, see the ReportSmith documentation.

Note *RecalcReport* will fail if the *Busy* property is *True*.

See also *Busy* property, *SetVariable* method, *SetVariableLines* method

Run

Run tells *ReportSmith* to preview or print the report.

```
procedure Run;
```

Description Call *Run* to preview or print the report specified by *ReportName*. If *Preview* is *True*, the report will be opened in a preview window. If *Preview* is *False*, the report will be printed. *Run* sends the *InitialValues* to ReportSmith before running the report.

The following code allows the user to specify the report with the Open dialog box and then runs the report.

```
if OpenDialog1.Execute then
begin
  Report1.ReportName := OpenDialog1.FileName;
  Report1.Run;
end;
```

When the report is being previewed by the user, *TReport* can tell ReportSmith to print the report using the *Print* method.

Note *Run* will fail if the *Busy* property is *True*.

See also *Busy* property, *InitialValues* property, *Preview* property, *Print* method, *ReportName* property

RunMacro

RunMacro runs a ReportBasic macro

```
function RunMacro(Macro: PChar): Boolean;
```

Description Use *RunMacro* to tell ReportSmith to run the ReportBasic macro named by the *Macro* parameter. The *RunMacro* method sends a message to ReportSmith to run the specified macro and looks for a confirmation message from ReportSmith in return. If *RunMacro* returns *True*, the message was sent successfully to ReportSmith. If it returns *False*, ReportSmith could not receive the message at the current time.

For information about ReportBasic macros, refer to the ReportSmith documentation.

Note *Macro* is the name of the macro, not the actual ReportBasic commands.

Note *RunMacro* will fail if the *Busy* property is *True*.

See also *Busy* property

SetVariable

SetVariable changes the value of a report variable.

```
function SetVariable(Name, Value: string): Boolean;
```

Description Call *SetVariable* to change the value of a report variable. The *Name* parameter specifies the report variable to change, and the *Value* parameter specifies the new value. If the *Value* for the parameter is greater than 255 characters, use *SetVariableLines*. Once the *SetVariable* method has been called, call the *RecalcReport* method to recalculate the report using the new value for the report variable.

The *SetVariable* method sends a message to ReportSmith to change the specified report variable with the new value, and looks for confirmation from ReportSmith in return. If *SetVariable* returns *True*, the message was received by ReportSmith successfully. If it returns *False*, ReportSmith could not receive the message or set the variable at the current time.

For information about report variables, see the ReportSmith documentation.

Note Before calling *SetVariable*, the report must be loaded. Load a report by calling the *Run* method.

Note *SetVariable* will fail if the *Busy* property is *True*.

See also *Busy* property, *InitialValues* property, *RecalcReport* method, *Run* method, *SetVariableLines* method

SetVariableLines

SetVariableLines changes the value of a report variable.

```
function SetVariableLines(Name, Value: TStrings): Boolean;
```

Description Call *SetVariableLines* to change the value of a report variable. The *Name* parameter specifies the report variable to change, and the *Value* parameter specifies the new value. Once the *SetVariable* method has been called, call the *RecalcReport* method to recalculate the report using the new value for the report variable.

The *SetVariableLines* method sends a message to ReportSmith to change the specified report variable with the new value, and looks for confirmation from ReportSmith in return. If *SetVariableLines* returns *True*, the message was received by ReportSmith successfully. If it returns *False*, ReportSmith could not receive the message or set the variable at the current time.

For information about report variables, see the ReportSmith documentation.

Note Before calling *SetVariableLines*, the report must be loaded. Load a report by calling the *Run* method.

Note *SetVariableLines* will fail if the *Busy* property is *True*.

See also *Busy* property, *InitialValues* property, *RecalcReport* method, *Run* method, *SetVariable* method

TRichEdit

Unit: ComCtrls Palette: *Win32*

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomEdit • TCustomMemo • TCustomRichEdit • TRichEdit

TRichEdit is a wrapper for a Windows rich text edit control.

Description Use a *TRichEdit* object to put a standard Windows rich text edit control on a form. Rich text edit controls let the user enter text that includes variation in font attributes and paragraph formatting information.

TRichEdit provides the properties and methods to enter and work with rich text. It does not provide any user interface components to make these formatting options available to users. Applications must implement the user interface components to surface the rich text capabilities of a *TRichEdit* object.

TRichEdit publishes many of the properties inherited from *TCustomRichEdit*, but does not introduce any new behavior.

See also *TDBMemo object*, *TMemo* object, *TParaAttributes* object, *TTextAttributes* object

Properties

The following table lists the properties for *TRichEdit*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
Alignment	published	taLeftJustify	TCustomMemo
BorderStyle	published	bsSingle	TCustomEdit
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent

Property	Visibility	Default	Documented in
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DefAttributes	public		TCustomRichEdit
DefaultConverter	public		TCustomRichEdit
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
HideScrollBars	published	True	TCustomRichEdit
HideSelection	published	True	TCustomRichEdit
Hint	published		TControl
ImeMode	published	imDontCare	TWinControl
ImeName	published		TWinControl
Left	published		TControl
Lines	published		TCustomRichEdit
MaxLength	published	0	TCustomEdit
Modified	public		TCustomEdit
Name	published		TControl
Owner	public, read-only		TComponent
PageRect	public		TCustomRichEdit
Paragraph	public, read-only		TCustomRichEdit
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PlainText	published	False	TCustomRichEdit
PopupMenu	published		TControl
ReadOnly	published	False	TCustomEdit
ScrollBars	published	ssNone	TCustomMemo

Property	Visibility	Default	Documented in
SelAttributes	public		TCustomRichEdit
SelLength	public		TCustomRichEdit
SelStart	public		TCustomRichEdit
SelText	public		TCustomRichEdit
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent
Text	public		TControl
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
WantReturns	published	True	TCustomMemo
WantTabs	published	False	TCustomMemo
Width	published		TControl
WindowProc	public		TControl
WordWrap	published	True	TCustomMemo

Events

The following table lists the events for *TRichEdit*:

Event	Visibility	Documented in
OnChange	published	TCustomEdit
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnProtectChange	published	TCustomRichEdit
OnResizeRequest	published	TCustomRichEdit
OnSaveClipboard	published	TCustomRichEdit
OnSelectionChange	published	TCustomRichEdit
OnStartDrag	published	TControl

Methods

The following table lists the methods for *TRichEdit*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TCustomRichEdit
ClearSelection	public	TCustomEdit
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
CopyToClipboard	public	TCustomEdit
Create	public	TCustomRichEdit
CreateParented	public	TWinControl
CutToClipboard	public	TCustomEdit
DefaultHandler	public	TObject
Destroy	public	TCustomRichEdit
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
FindText	public	TCustomRichEdit
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent

Method	Visibility	Documented in
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetSelTextBuf	public	TCustomRichEdit
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
PasteFromClipboard	public	TCustomEdit
Perform	public	TControl
Print	public	TCustomRichEdit
Realign	public	TWinControl
Refresh	public	TControl
RegisterConversionFormat	public	TCustomRichEdit
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SelectAll	public	TCustomEdit
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetSelTextBuf	public	TCustomEdit
SetTextBuf	public	TControl

Method	Visibility	Documented in
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

TSaveDialog

Unit: Dialogs | Palette: Dialogs

TObject • TPersistent • TComponent • TCommonDialog • TOpenDialog • TSaveDialog

TSaveDialog displays a Save As dialog for saving files.

Description *TSaveDialog* displays a modal Windows dialog box for selecting file names and saving files. The dialog does not appear at runtime until it is activated by a call to the *Execute* method. When the user clicks Save, the dialog closes and the selected file name is stored in the *FileName* property.

See also *TSavePictureDialog* object

Properties

The following table lists the properties of *TSaveDialog*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Ctl3D	published	True	TCommonDialog
DefaultExt	published		TOpenDialog
DesignInfo	public		TComponent
FileEditStyle	published		TOpenDialog
FileName	published		TOpenDialog
Files	public, read-only		TOpenDialog
Filter	published		TOpenDialog
FilterIndex	published	1	TOpenDialog
HelpContext	published	0	TCommonDialog
HistoryList	public		TOpenDialog
InitialDir	published		TOpenDialog
Name	published		TComponent
Options	published	[]	TOpenDialog
Owner	public, read-only		TComponent

S

Property	Visibility	Default	Documented in
Tag	published	0	TComponent
Title	published		TOpenDialog
VCLComObject	public		TComponent

Events

The following table lists the events of *TSaveDialog*:

Event	Visibility	Documented in
OnClose	published	TCommonDialog
OnSelectionChange	published	TOpenDialog
OnShow	published	TCommonDialog

Methods

The following table lists the methods of *TSaveDialog*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TOpenDialog
DefaultHandler	public	TObject
Destroy	public	TOpenDialog
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
Execute	public	TSaveDialog
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject

Method	Visibility	Documented in
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetStaticRect	protected	TOpenDialog
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Execute

Execute displays the Save As dialog box.

function Execute: Boolean; **override**;

Description *Execute* opens the Save As dialog, returning *True* when the user selects a file name and clicks Save.

TSavePictureDialog

Unit: ExtDlgs Palette: Dialogs

TObject • TPersistent • TComponent • TCommonDialog • TOpenDialog • TOpenPictureDialog • TSavePictureDialog

TSavePictureDialog displays a Save As dialog for saving graphics files.

Description *TSavePictureDialog* displays a modal Windows dialog box for selecting file names and saving graphics files. This component is just like *TSaveDialog*, except that it includes a rectangular preview region. If the selected image can be read by *TPicture*, it is displayed in the preview region; supported file types include bitmap (.BMP), icon (.ICO), Windows metafile (.WMF), and enhanced Windows metafile (.EMF). If the selected image cannot be displayed, "(None)" appears in the preview region.

See also *TOpenPictureDialog* object, *TPicture* object, *TSaveDialog* object

Properties

The following table lists the properties of *TSavePictureDialog*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Ctl3D	published	True	TCommonDialog
DefaultExt	published		TOpenDialog
DesignInfo	public		TComponent
FileEditStyle	published		TOpenDialog
FileName	published		TOpenDialog
Files	public, read-only		TOpenDialog
Filter	published		TOpenDialog
FilterIndex	published	1	TOpenDialog
HelpContext	published	0	TCommonDialog
HistoryList	public		TOpenDialog
InitialDir	published		TOpenDialog
Name	published		TComponent
Options	published	[]	TOpenDialog
Owner	public, read-only		TComponent
Tag	published	0	TComponent
Title	published		TOpenDialog
VCLComObject	public		TComponent

Events

The following table lists the events of *TSavePictureDialog*:

Event	Visibility	Documented in
OnClose	published	TCommonDialog
OnSelectionChange	published	TOpenDialog
OnShow	published	TCommonDialog

Methods

The following table lists the methods of *TSavePictureDialog*:ss

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject

Method	Visibility	Documented in
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TOpenPictureDialog
DefaultHandler	public	TObject
Destroy	public	TOpenPictureDialog
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
Execute	public	TSavePictureDialog
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Execute

Execute displays the Save As dialog box.

```
function Execute: Boolean; override;
```

Description *Execute* opens the Save As dialog, returning *True* when the user selects a file name and clicks Save.

TScreen

Unit: Forms

TObject • TPersistent • TComponent • TScreen

Description

TScreen represents the state of the screen in which an application runs.

TScreen introduces properties that keep track of

- What forms and data modules have been instantiated by the application.
- The active form, and the active control within that form.
- The size and resolution of the screen.
- The cursors and fonts available for the application to use.

There is a global variable, *Screen*, of type *TScreen*, that is already instantiated for use by Delphi applications. Use *Screen* to obtain information about the current state of the screen in an application.

See also

Screen variable, *TApplication* object, *TDataModule* object, *TForm* object, *TFont* object

Properties

The following table lists the properties of *TScreen*:

Property	Visibility	Default	Documented in
ActiveControl	public, read-only		TScreen
ActiveCustomForm	public, read-only		TScreen
ActiveForm	public, read-only		TScreen
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Cursor	public		TScreen
Cursors	public		TScreen
CustomFormCount	public, read-only		TScreen
CustomForms	public, read-only		TScreen
DataModuleCount	public, read-only		TScreen
DataModules	public, read-only		TScreen
DefaultIme	public, read-only		TScreen
DefaultKbLayout	public, read-only		TScreen
DesignInfo	public		TComponent

Property	Visibility	Default	Documented in
Fonts	public, read-only		TScreen
FormCount	public, read-only		TScreen
Forms	public, read-only		TScreen
Height	public, read-only		TScreen
IconFont	public		TScreen
Imes	public, read-only		TScreen
Name	published		TComponent
Owner	public, read-only		TComponent
PixelsPerInch	public, read-only		TScreen
Tag	published	0	TComponent
VCLComObject	public		TComponent
Width	public, read-only		TScreen

ActiveControl

ActiveControl indicates which control currently has input focus on the screen.

```
property ActiveControl: TWinControl;
```

Description Read *ActiveControl* to learn which windowed control object in the active form currently receives the input from the keyboard.

ActiveControl is a read-only property. To change the *ActiveControl*, use the *SetFocusedControl* method of the form which contains control that should receive focus.

After focus shifts from one control to another, the screen receives an *OnActiveControlChange* event.

See also *OnActiveControlChange* event, *TForm.ActiveControl* property, *TForm.SetFocusedControl* method

ActiveCustomForm

ActiveCustomForm indicates the descendant of *TCustomForm* that currently has focus.

```
property ActiveCustomForm: TCustomForm;
```

Description Read *ActiveForm* to learn which form or property page has input focus. If the active custom form is a *TForm* object, *ActiveCustomForm* is the same as *ActiveForm*.

See also *ActiveForm* property, *OnActiveFormChange* event, *TForm* object, *TPropertyPage* object

ActiveForm

ActiveForm indicates which form currently has focus.

property ActiveForm: TForm;

Description Read *ActiveForm* to learn which form in the application has input focus. If the application is not currently active, *ActiveForm* is the form that will have focus when the application becomes active again.

ActiveForm is a read-only property. To change the *ActiveForm*, use the *SetFocus* method of the form that should receive focus. *ActiveForm* will also change if the *SetFocusedControl* method of an inactive form is called to set focus to a control on that inactive form.

After focus shifts from one form to another, the screen receives an *OnActiveFormChange* event.

See also *ActiveCustomForm* property, *OnActiveFormChange* event, *TForm.ActiveForm* property, *TForm.SetFocus* method, *TForm.SetFocusedControl* method

Cursor

Cursor controls the mouse cursor image on a global level.

property Cursor: TCursor;

Description Set *Cursor* to override the cursor image associated with individual control objects. When *Cursor* is *crDefault*, the individual objects determine the cursor image. Assigning any other value sets the mouse cursor image for all windows belonging to the application. The global mouse cursor image remains in effect until the screen's *Cursor* property is changed back to *crDefault*.

Assignments to the Screen object's cursor property are typically guarded by a **try...finally** statement to ensure that normal cursor behavior is restored. For example:

```
Screen.Cursor := crHourglass;                { Show hourglass cursor }
try
  { Do some lengthy operation }
finally
  Screen.Cursor := crDefault;                 { Always restore to normal }
end;
```

Cursor can be set to any of the cursor values available in the *Cursors* property. This can be one of the built-in cursor values, or a custom cursor that has been added to the *Cursors* property.

See also *Cursors* property, *TControl.Cursor* property

Cursors

Cursors is an array of cursors available to the application.

property Cursors[Index: Integer]: HCursor;

Description Use *Cursor* to access a particular cursor for use by the application or by a control within the application. *TScreen* includes several built-in cursors that are indexed by symbolic cursor constants. The image associated with the built-in cursors constants can be changed by setting the *Cursors* property.

These are the cursor constants and their position in the *Cursors* property array

Constant	Value	Image
crDefault	0	depends on the *ActiveControl*.
crNone	–1	
crArrow	–2	
crCross	–3	
crIBeam	–4	
crSize	–5	
crSizeNESW	–6	
crSizeNS	–7	
crSizeNWSE	–8	
crSizeWE	–9	
crUpArrow	–10	
crHourGlass	–11	
crDrag	–12	

Constant	Value	Image
crNoDrop	–13	
crHSplit	–14	
crVSplit	–15	
crMultiDrag	–16	
crSQLWait	–17	SQL
crNo	–18	
crAppStart	–19	
crHelp	–20	?
crHandPoint	–21	

Custom cursors can be added to the *Cursors* property for use by the application or any of its controls. To add a custom cursor to an application:

1 Create the cursor resource using a resource editor.

2 Declare a cursor constant with a value that does not conflict with an existing cursor constant.

3 Use the Windows API function *LoadCursor* to obtain a handle to the new cursor.

4 Set the *Cursors* property, indexed by the newly declared cursor constant, to the handle obtained from LoadCursor.

For example, assume that a custom cursor has been added to the resources of an application with the name *NewCursor*. The following code makes this cursor available to the application via the constant *crMyCursor*, and sets it as the global cursor to the application.

```
const
  crMyCursor = 5;
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  Screen.Cursors[crMyCursor] := LoadCursor(HInstance, 'NewCursor');
  Cursor := crMyCursor;
end;
```

Note Don't call the Windows API function *DestroyCursor* when finished with the custom cursor; Delphi does this automatically.

See also *Cursor* property, *TControl.Cursor* property

CustomFormCount

CustomFormCount is the number of forms or property pages displayed on the screen.

property CustomFormCount: Integer;

Description Read *CustomFormCount* to learn the number of forms or property pages currently displayed on the screen. These forms can be accessed by the *CustomForms* property. *CustomFormCount* can be used with *CustomForms* to iterate over all the forms and property pages in an application.

See also *CustomForms* property, *FormCount* property

CustomForms

CustomForms is a list of all the forms and property pages that are currently displayed in the application.

property CustomForms[Index: Integer]: TCustomForm;

Description Use *CustomForms* to access a form or property page by index. The value of *Index* is a number between zero (the first custom form) and *CustomFormCount* –1. *CustomForms* can be used with *CustomFormCount* when an application needs to iterate over all its forms and property pages.

For example, the following code adds the name of all forms and property pages on the screen to *ListBox1* when *Button1* is clicked.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  For I := 0 to Screen.CustomFormCount-1 do
    ListBox1.Items.Add(Screen.CustomForms[I].Name);
end;
```

See also *CustomFormCount* property, *Forms* property

DataModuleCount

DataModuleCount is the number of data modules that are currently instantiated in the application.

```
property DataModuleCount: Integer;
```

Description Read *DataModuleCount* to learn the number of data modules in the *DataModules* property. *DataModuleCount* can be used with *DataModules* when an application needs to iterate over all its data modules.

See also *DataModules* property, *TDataModule* object

DataModules

DataModules is a list of all the data modules that are currently instantiated in the application.

```
property DataModules[Index: Integer]: TDataModule;
```

Description Use *DataModules* to access an instantiated data module by index. The value of *Index* is a number between zero (the first data module) and *DataModuleCount* –1. *DataModules* can be used with *DataModuleCount* when an application needs to iterate over all its data modules.

See also *DataModuleCount* property, *TDataModule* object

DefaultIme

DefaultIme is the name of the input method editor (IME) that is active when the *TScreen* object is created.

```
property DefaultIme: string;
```

Description In applications that support asian language characters, read *DefaultIme* to determine the IME that is active when the application starts.

An IME is a front-end input processor for asian language characters. The IME hooks all keyboard input, converts it to asian characters in a conversion window, and sends the converted characters or strings on to the Delphi application. There can be multiple IMEs installed on a single system. They vary in user interface, conversion dictionaries, and input method.

Individual controls can override the default IME. When a control specifies a different IME should be used to enter information into the control, the IME that processes keyboard input changes when the control has focus. When the control loses focus, *DefaultIme* is restored as the system IME.

See also *DefaultKbLayout* property, *Imes* property, *TWinControl.ImeName* property, *TWinControl.ResetIme* method

DefaultKbLayout

DefaultKbLayout is the Windows handle to the keyboard layout that was active when the application started running.

```
property DefaultKbLayout: HKL;
```

Description Use *DefaultKbLayout* for making Windows API calls that require a handle to a keyboard layout. For example, if an application changes the keyboard layout to match the settings for another country, the value of *DefaultKbLayout* can be used as a parameter to the Windows *ActivateKeyboardLayout* function in order to restore the original layout.

For asian keyboards, the keyboard layout identifies the input method editor (IME) as well as the default language. Use *DefaultKbLayout* in Windows API calls that work with the IME. *DefaultKbLayout* is the keyboard layout handle associated with the IME named by the *DefaultIme* property.

See also *DefaultIme* property, *Imes* property

Fonts

Fonts is a list of face names for all fonts supported by the screen.

```
property Fonts: TStrings;
```

Description Read *Fonts* to learn what screen fonts are currently installed. Applications can use *Fonts* to ensure that they do not request a font that is not installed on the user's system. When an application uses a *TFont* object to request a font that is not installed, Windows substitutes another font, which may not be a font appropriate to the application's needs.

Note *Fonts* is a list of screen fonts. It does not give any indication of the printer fonts available to the application.

See also *TFont* object

FormCount

FormCount is the number of forms displayed on the screen.

```
property FormCount: Integer;
```

Description Read *FormCount* to learn the number of forms currently displayed on the screen. These forms can be accessed by the *Forms* property. *FormCount* can be used with *Forms* to iterate over all the forms in an application.

See also *Forms* property, *TForm* object

Forms

Forms is a list of all the forms that are currently displayed in the application.

```
property Forms[Index: Integer]: TForm;
```

Description Use *Forms* to access a form by index. The value of *Index* is a number between zero (the first form) and *FormCount* – 1. *Forms* can be used with *FormCount* when an application needs to iterate over all its forms, including all dialogs.

For example, the following code adds the name of all forms on the screen to *ListBox1* when *Button1* is clicked.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  For I := 0 to Screen.FormCount-1 do
    ListBox1.Items.Add(Screen.Forms[I].Name);
end;
```

See also *FormCount* property, *TForm* object

Height

Height is the vertical size of the screen in pixels.

property Height: Integer;

Description Read *Height* to learn the size of the user's screen in pixels. The size or position of objects placed on the screen, can then be set to ensure that the objects fit on screen and don't look crowded. For example, the following code determines the height of all forms on the screen and resizes any that are taller than the screen height.

```
with Screen do
  for I := 0 to FormCount-1 do
  if Forms[I].Height > Height than Forms[I].Height := Height;
```

See also *PixelsPerInch* property, *TControl.Height* property, *Width* property

IconFont

IconFont is the font used to display the labels on the icons in file selection dialogs.

property IconFont: TFont;

Description *IconFont* stores the Windows 95 icon font setting. This font is used to label the icons on the Windows desktop, and icons in the file selection dialogs.

See also *Fonts* property, *TFont* object

Imes

Imes is a list of all input method editors (IMEs) installed on the system.

property Imes: TStrings;

Description Read *Imes* to obtain a list of all IMEs that are listed in the Windows registry. An IME is a front-end input processor for asian language characters. The IME hooks all keyboard input, converts it to asian characters in a conversion window, and sends the converted characters or strings on to the Delphi application. IMEs vary in user interface, conversion dictionaries, and input method.

Individual controls can require a specific IME to process the keyboard input when they have focus. To specify an IME, the control sets its *ImeName* property to one of the strings in *Imes*.

Each IME name in *Imes* has an associated keyboard layout handle. A keyboard layout handle is a Windows handle that identifies the primary language and IME used to process keyboard input. Use the keyboard layout handle in Windows API calls for programming the IME. To get the keyboard layout handle for a particular IME, a control can use code such as the following:

```
HKeyboard := Screen.Imes.Objects(Screen.Imes.IndexOf(ImeName));
```

See also *DefaultIme* property, *DefaultKbLayout* property, *TStrings* object, *TWinControl.ImeName* property

PixelsPerInch

PixelsPerInch is the number of screen pixels that make up a logical inch in the vertical direction.

```
property PixelsPerInch: Integer;
```

Description Read *PixelsPerInch* to convert between measurements in pixels and measurements in logical inches. *PixelsPerInch* is only accurate for vertical measurements. Most screens have a different scaling factor for horizontal measurements. The value in *PixelsPerInch* is set from Windows when Delphi loads.

See also *Height* property, *TForm.PixelsPerInch* property

Width

Width is the horizontal size of the screen in pixels.

```
property Width: Integer;
```

Description Read *Width* to learn the size of the user's screen in pixels. The size or position of objects placed on the screen, can then be set to ensure that the objects fit on screen and don't look crowded. For example, the following code determines the width of all forms on the screen and resizes any that are wider than the screen width.

```
with Screen do
  for I := 0 to FormCount-1 do
  if Forms[I].Width > Width than Forms[I].Width := Width;
```

See also *Height* property, *TControl.Width* property

S

Events

The following table lists the events of *TScreen*:

Event	Visibility	Documented in
OnActiveControlChange	public	TScreen
OnActiveFormChange	public	TScreen

OnActiveControlChange

OnActiveControlChange occurs immediately after input focus changes to a new windowed control.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnActiveControlChange: TNotifyEvent;
```

Description Write an *OnActiveControlChange* event handler to take specific action when input focus changes to a new control. The change in focus may be within the active form, or across forms to a new form that then becomes the active form.

When focus moves from one control to another, the following events occur.

1. If the new focused control is in a different form, focus moves to the new form.
2. Focus moves to the new active control.
3. If the active form changed, an *OnActiveFormChange* event occurs.
4. An *OnActiveControlChange* event occurs.

See also *ActiveControl* property, *OnActiveFormChange* event, *TForm.SetFocusedControl* method

OnActiveFormChange

OnActiveFormChange occurs immediately after a new form becomes active in a multi-form application.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnActiveFormChange: TNotifyEvent;
```

Description Write an *OnActiveFormChange* event handler to take specific action when a new form becomes active. *OnActiveFormChange* occurs when the active form for the application changes, not when a form becomes active because the application becomes active.

See also *ActiveForm* property, *OnActiveControlChange* event, *TForm.SetFocus* method

Methods

The following table lists the methods of *TScreen*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TScreen
DefaultHandler	public	TObject
Destroy	public	TScreen
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

S

Create

Create creates an instance of *TScreen*.

constructor Create(AOwner: TComponent); **override**;

Description Do not create separate instances of *TScreen*. There is a global variable, *Screen*, of type *TScreen*, that is already instantiated for use by Delphi applications. The properties of *Screen* are global to the entire application.

Create allocates memory for the *TScreen* object and

- Fills the *Cursors* property with the built-in cursors.
- Fills the *Fonts* property with the available screen fonts.
- Fills the *Imes* property will all registered input method editors (IMEs), and sets the *DefaultIme* and *DefaultKbLayout* properties.
- Sets the *PixelsPerInch* property for the current screen resolution.
- Creates helper objects for the *IconFont*, *Forms* and *DataModules* properties.

See also *Cursors* property, *DataModules* property, *DefaultIme* property, *DefaultKbLayout* property, *Fonts* property, *Forms* property, *IconFont* property, *Imes* property, *PixelsPerInch* property

Destroy

Destroy destroys an instance of *TScreen*.

destructor Destroy; **override**;

Description Do not destroy the global object *Screen* that is automatically instantiated in Delphi applications. *Screen* is used by many Delphi objects.

Destroy frees the memory associated with the *DataModules*, *Forms*, *Fonts*, *IconFont* and *Imes* properties, and releases the cursors used by the *Cursors* property.

See also *Cursors* property, *DataModules* property, *Fonts* property, *Forms* property, *IconFont* property, *Imes* property

TScrollBar

Unit: StdCtrls | Palette: Standard

TObject • TPersistent • TComponent • TControl • TWinControl • TScrollBar

TScrollBar is a Windows scroll bar, which is used to scroll the contents of a window, form, or control.

Description Use *TScrollBar* to add a free-standing scrollbar to a form. Many controls have properties that add a scroll bars which are an integral part of the control. *TScrollBar* allows groupings of controls or controls that do not have integrated scroll bars to be scrolled when the user manipulates the *TScrollBar* object.

See also *TScrollingWinControl* object, *TTrackBar* object, *TUpDown* object

Properties

The following table lists the properties of *TScrollBar*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Kind	published	sbHorizontal	TScrollBar
LargeChange	published	1	TScrollBar
Left	published		TControl
Max	published	100	TScrollBar
Min	published	0	TScrollBar
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentCtl3D	published	True	TWinControl

Property	Visibility	Default	Documented in
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
Position	published	0	TScrollBar
ShowHint	published		TControl
Showing	public, read-only		TWinControl
SmallChange	published	1	TScrollBar
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Kind

Kind specifies whether the scroll bar is horizontal or vertical.

```
property Kind: TScrollBarKind;
```

Description Set *Kind* to indicate the orientation of the scrollbar. These are the possible values:

Value	Meaning
sbHorizontal	Scroll bar is horizontal
sbVertical	Scroll bar is vertical

LargeChange

LargeChange determines how much *Position* changes when the user clicks the scroll bar on either side of the thumb or presses *PgUp* or *PgDn*.

```
type TScrollBarInc = 1..32767;

property LargeChange: TScrollBarInc;
```

Description Set *LargeChange* to specify how much to change the value of *Position* when the user clicks on the side of the thumb. The *Max* and *Min* properties define the total range over which *Position* can vary.

For example, if *Max* is 30000 and *Min* is 0, the scroll box can assume 30,000 positions. If the *LargeChange* property setting is 10000 and the *Position* property is 0, the user can click the scroll bar track three times after the thumb before the thumb is moved all the way to the end of the scroll bar (30000 / 10000 = 3).

LargeChange should correspond to the portion of this range that the user sees in one page or screen, expressed in the logical units used by the *Max*, *Min*, and *Position* properties.

See also *Max* property, *Min* property, *Position* property, *SmallChange* property

Max

Max is the maximum position represented by the scrollbar.

property Max: Integer;

Description Set *Max* to the maximum value the *Position* property can take. The *Max* and *Min* properties define the total range over which *Position* can vary.

See also *LargeChange* property, *Min* property, *Position* property, *SetParams* method, *SmallChange* method

Min

Min is the minimum position represented by the scrollbar.

property Min: Integer;

Description Set *Min* to the minimum value the *Position* property can take. The *Min* and *Max* properties define the total range over which *Position* can vary.

See also *LargeChange* property, *Max* property, *Position* property, *SetParams* method, *SmallChange* method

Position

Position is the current position of the scrollbar.

property Position: Integer;

Description Read *Position* to determine the current position of the thumb. This value can be used to determine how to scroll any components controlled by the scroll bar. When the user scrolls the scroll bar, the value of *Position* changes. Set *Position* to programmatically move the thumb of the scroll bar.

The number of possible positions on the scroll bar is determined by the difference between the *Max* property and the *Min* property. If *Position* has the same value as *Min*, the thumb appears at the far left of a horizontal scroll bar or the top of a vertical scroll bar. If *Position* has the same value as *Max*, the thumb appears at the far right of a horizontal scroll bar or the bottom of a vertical scroll bar.

See also *Max* property, *Min* property, *SetParams* method

SmallChange

SmallChange determines how much *Position* changes when the user clicks the arrow buttons on the scroll bar or presses the arrow keys on the keyboard.

```
type TScrollBarInc = 1..32767;

property SmallChange: TScrollBarInc;
```

Description Set *SmallChange* to specify how much to change the value of *Position* when the user clicks on the arrow buttons. The *Max* and *Min* properties define the total range over which *Position* can vary.

For example, if *Max* is 100 and *Min* is 0, the scroll box can assume 100 positions. If the *SmallChange* property setting is 5 and the *Position* property is 0, the user can click the down or right arrow button 20 times before the thumb is moved all the way to the end of the scroll bar.

SmallChange should correspond to the portion of this range that the user sees as one line or one item, expressed in the logical units used by the *Max*, *Min*, and *Position* properties.

See also *LargeChange* property, *Max* property, *Min* property, *Position* property

Events

The following table lists the events of *TScrollBar*:

Event	Visibility	Documented in
OnChange	published	TScrollBar
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnScroll	published	TScrollBar
OnStartDrag	published	TControl

OnChange

OnChange occurs immediately after the value of *Position* changes.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnChange: TNotifyEvent;
```

Description Write an *OnChange* event handler to control another object or group of objects using *TScrollBar*. Adjust the objects controlled by the scrollbar to reflect the new value of *Position*.

OnChange occurs whether the value of *Position* is changed by the user or programmatically. If the value of *Position* is changed by the user, *OnChange* occurs immediately after the *OnScroll* event.

See also *OnScroll* event, *Position* property

OnScroll

OnScroll occurs when the user scrolls the scroll bar with the mouse or keyboard.

```
type
  TScrollCode = (scLineUp, scLineDown, scPageUp, scPageDown, scPosition, scTrack, scTop,
    scBottom, scEndScroll);
  TScrollEvent = procedure(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer)
    of object;

property OnScroll: TScrollEvent;
```

Description Write an *OnScroll* to adjust the value of *Position* when the user scrolls the scroll bar. The *ScrollPos* parameter contains the new value that is about to be assigned to the *Position* property as a result of the scroll. Change *ScrollPos* to alter the result of the scroll.

The *ScrollCode* parameter indicates the type of user action that just scrolled the scrollbar. The *ScrollCode* parameter is one of these values:

Value	Meaning
scLineUp	User clicked the top or left scroll arrow or pressed the *Up* or *Left* arrow key.
scLineDown	User clicked the bottom or right scroll arrow or pressed the *Down* or *Right* arrow key.
scPageUp	User clicked the area to the left of the thumb tab or pressed the *PgUp* key.
scPageDown	User clicked the area to the right of the thumb tab or pressed the *PgDn* key.
scPosition	User positioned the thumb tab and released it.
scTrack	User is moving the thumb tab.
scTop	User moved the thumb tab to the top or far left on the scroll bar.
scBottom	User moved the thumb tab to the bottom or far right on the scroll bar.
scEndScroll	User finished moving the thumb tab on the scroll bar.

See also *LargeChange* property, *OnChange* event, *Position* property, *SmallChange* property

S

Methods

The following table lists the methods of *TScrollBar*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TScrollBar
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TWinControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent

Method	Visibility	Documented in
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetParams	public	TScrollBar
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

S

Create

Create creates and initializes an instance of *TScrollBar*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to create an instance of *TScrollBar* at runtime. Scroll bars placed on a form at design time are created automatically. *Create* calls the inherited *Create* method, then sets the initial values for the scroll bar as follows:

- *ControlStyle* is set to [*csFramed, csDoubleClicks*]
- *Height* is set to the system horizontal scrollbar height, and *Width* to 121.
- *Kind* is set to *sbHorizontal*.
- *Position* is set to 0, *Min* to 0, and *Max* to 100.
- *SmallChange* and *LargeChange* are both set to 1.
- *TabStop* is set to *True*.

See also *ControlStyle* property, *Height* property, *Kind* property, *LargeChange* property, *Max* property, *Min* property, *Position* property, *SmallChange* property, *TabStop* property, *TWinControl.Create* method, *Width* property

SetParams

SetParams sets the *Position*, *Max*, and *Min* property values all at once.

```
procedure SetParams(APosition, AMax, AMin: Integer);
```

Description Use the *SetParams* method to set the *Position*, *Max*, and *Min* property values all at once, preventing repaints that would occur if each property value were set separately.

See also *Max* property, *Min* property, *Position* property

TScrollBox

Unit: Forms | Palette: Additional

TObject • TPersistent • TComponent • TControl • TWinControl • TScrollingWinControl • TScrollBox

TScrollBox is the component for a scrolling area (a scroll box) in a window.

Description Use *TScrollBox* to create a scroll box in a window.

One use of a scroll box is to prevent areas of a window, such as a toolbar or status bar (*TPanel* components), from scrolling. To prevent a toolbar and status bar from scrolling, hide the window's scroll bars, and then position a scroll box in the client area of the window between the toolbar and status bar. The scroll bars associated with the scroll box will appear to belong to the window, but will scroll only the area inside the scroll box.

Another use of scroll boxes is to create multiple scrolling areas (views) in a window. Views are common in commercial word-processor, spreadsheet, and project management applications.

A scroll box can contain objects, such as *TButton* and *TCheckBox* objects.

See also *TForm* object

Properties

The following table lists the properties of *TScrollBox*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
AutoScroll	published	True	TScrollingWinControl
BorderStyle	published	bsSingle	TScrollBox
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
HorzScrollBar	published		TScrollingWinControl
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl

Property	Visibility	Default	Documented in
ParentColor	published	True	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
VertScrollBar	published		TScrollingWinControl
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

BorderStyle

BorderStyle is a value that specifies whether the scroll box has a border.

property BorderStyle: TBorderStyle;

Description Use *BorderStyle* to get or set whether the scroll box has a border. *BorderStyle* can be one of the following *TBorderStyle* values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

Events

The following table lists the events of *TScrollBox*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl

Event	Visibility	Documented in
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnResize	published	TScrollBox

OnResize

OnResize occurs when the scroll box is resized.

```
type TNotifyEvent = procedure(Sender: TObject) of object;
property OnResize: TNotifyEvent;
```

Description Use *OnResize* to perform special processing when the scroll box is resized.

Methods

The following table lists the methods of *TScrollBox*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TScrollBox
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TScrollingWinControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
DisableAutoRange	public	TScrollingWinControl
Dispatch	public	TObject

Method	Visibility	Documented in
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EnableAutoRange	public	TScrollingWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl

Method	Visibility	Documented in
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
ScrollInView	public	TScrollingWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

Create

Create creates and initializes a new *TScrollBox* object.

```
constructor Create(AOwner: TComponent); override;
```

Description Use *Create* to create and initialize a new *TScrollBox* object.

AOwner is a *TComponent* object that encapsulates the parent object for the scroll box.

Create performs the following tasks:

- Calls the inherited *Create* method, passing it *AOwner*.
- Sets the *ControlStyle* property to [*csAcceptsControls, csCaptureMouse, csClickEvents, csSetCaption, csDoubleClicks*].
- Sets the *Width* property to 185.
- Sets the *Height* property to 41.
- Sets the *BorderStyle* property to bsSingle.

See also *BorderStyle* property, *ControlStyle* property, *Height* property, *TScrollingWinControl.Create* method, *Width* property

S

TScrollingWinControl

Unit: Forms

TObject • TPersistent • TComponent • TControl • TWinControl • TScrollingWinControl

TScrollingWinControl is a windowed control that supports scrolling.

Description Use *TScrollingWinControl* to create a windowed control that supports scrolling.

Typically, a scrolling windowed control has horizontal and vertical scroll bars and scrolls a child control into view when the child control receives focus.

Examples of scrolling windowed controls include *TForm* and *TScrollBox* objects.

See also *TForm object, TScrollBox object*

Properties

The following table lists the properties of *TScrollingWinControl*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
AutoScroll	protected	True	TScrollingWinControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
Enabled	public	True	TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
HorzScrollBar	published		TScrollingWinControl
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentWindow	public		TWinControl

Property	Visibility	Default	Documented in
ShowHint	public		TControl
Showing	public, read-only		TWinControl
TabOrder	public	–1	TWinControl
TabStop	public	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
VertScrollBar	published		TScrollingWinControl
Visible	public	True	TControl
Width	published		TControl
WindowProc	public		TControl

AutoScroll

AutoScroll is a Boolean property that specifies whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

```
property AutoScroll: Boolean;
```

Description If *AutoScroll* is *True*, the scroll bars appear automatically when necessary. For example, if the user resizes the scrolling windowed control so that some of its controls are partially obscured, scroll bars appear.

If *AutoScroll* is *False*, scroll bars don't appear automatically. In this case, use the *HorzScrollBar* and *VertScrollBar* properties to make scroll bars appear.

See also *AutoScrollInView method, HorzScrollBar property, VertScrollBar property*

HorzScrollBar

HorzScrollBar is a *TControlScrollBar* object that encapsulates the horizontal scroll bar for the scrolling windowed control.

```
property HorzScrollBar: TControlScrollBar;
```

Description Use *HorzScrollBar* to hide, show, or manipulate the horizontal scroll bar for the scrolling windowed control.

See also *AutoScroll property, TControlScrollBar component, VertScrollBar property*

VertScrollBar

VertScrollBar is a *TControlScrollBar* object that encapsulates the vertical scroll bar for the scrolling windowed control.

```
property VertScrollBar: TControlScrollBar;
```

Description *Use VertScrollBar* to hide, show, or manipulate the vertical scroll bar for the scrolling windowed control.

See also *AutoScroll property, HorzScrollBar* property, *TControlScrollBar* component

Methods

The following table lists the methods of *TScrollingWinControl*:

Method	Visibility	Documented in
AlignControls	protected	TScrollingWinControl
Assign	public	TPersistent
AutoScrollInView	protected	TScrollingWinControl
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ChangeScale	protected	TScrollingWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TScrollingWinControl
CreateParams	protected	TScrollingWinControl
CreateParented	public	TWinControl
CreateWnd	protected	TScrollingWinControl
DefaultHandler	public	TObject
Destroy	public	TScrollingWinControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
DisableAutoRange	public	TScrollingWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EnableAutoRange	public	TScrollingWinControl
EndDrag	public	TControl
FieldAddress	public	TObject

Method	Visibility	Documented in
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
ScrollInView	public	TScrollingWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl

Method	Visibility	Documented in
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

AlignControls

AlignControls aligns the controls in the scrolling windowed control.

```
procedure AlignControls(AControl: TControl; var ARect: TRect); override;
```

Description Use *AlignControls* to align the controls in the scrolling windowed control according to the values of their *Align* properties.

AControl is a *TControl* object specifying the control that takes precedence in alignment when other controls have the same *Align* value. *AControl* can be **nil**.

ARect is a *TRect* object specifying the area in the scrolling windowed control that contains the controls to align. *ARect* is automatically set to the entire area of the scrolling windowed control.

If *AutoScroll* is *True* and the scrolling windowed control is not being resized, the ranges of the scroll bars on the scrolling windowed control are automatically recalculated.

After recalculating the ranges of the scroll bars and setting the value of *ARect*, *AlignControls* calls *TWinControl.AlignControls*, passing the *AControl* and *ARect* objects.

See also *Align* property, *AutoScroll* property, *TControl* object, *TRect* type, *TWinControl.AlignControls* method

AutoScrollInView

AutoScrollInView scrolls a control into the visible area of the scrolling windowed control.

```
procedure AutoScrollInView(AControl: TControl)
```

Description Use *AutoScrollInView* to scroll a control into the visible area of the scrolling windowed control.

AControl is a *TControl* object encapsulating the control to scroll into the visible area of the scrolling windowed control.

Unlike *ScrollInView*, *AutoScrollInView* does not perform scrolling when the *ComponentState* property of *AControl* or the scrolling windowed control is *csLoading*.

AutoScrollInView calls *ScrollInView* to perform any needed scrolling.

See Also *ComponentState* property, *ScrollInView* method

ChangeScale

ChangeScale resizes the scrolling windowed control by a specific ratio.

```
procedure ChangeScale(M, D: Integer); override;
```

Description Use *ChangeScale* to resize the scrolling windowed control by a specific ratio.

M is the multiplier of the ratio; *D* is the divisor of the ratio. For example, to resize the scrolling windowed control to four-fifths of its current size, set *M* to four and *D* to five.

ChangeScale recalculates the ranges of the scroll bars on the scrolling windowed control according to the ratio, and then calls *TControl.ChangeScale*.

See also *TControl.ChangeScale* method

Create

Create is a method that creates and initializes a new *TScrollingWinControl* object.

```
constructor Create(AOwner: TComponent); override;
```

Description Use *Create* to create and initialize a new *TScrollingWinControl* object at runtime. Scrolling windowed controls placed on forms at design time are created automatically.

AOwner is a *TComponent* object that owns the scrolling windowed control.

Create performs the following tasks:

- Calls *TWinControl.Create* to create the *TScrollingWinControl* object and insert it into its owner.
- Creates horizontal and vertical scroll bar objects and assigns them to *HorzScrollBar* and *VertScrollBar* respectively.
- Sets *AutoScroll* to *True*.

See also *AutoScroll* property, *HorzScrollBar* property, *TComponent* object, *TWinControl.Create* method, *VertScrollBar* property

CreateParams

CreateParams modifies the style of the Window class so that the entire window is not redrawn when the width or height of the client area changes.

```
procedure CreateParams(var Params: TCreateParams); override;
```

Description *CreateParams* calls the inherited method and uses that Window class style, removing the CS_HREDRAW and CS_VREDRAW styles from it.

By removing these styles the form will only redraw the invalidated regions, not the entire window.

CreateWnd

CreateWnd is a method that creates a Windows control corresponding to the *TScrollingWinControl* object.

```
procedure CreateWnd; override;
```

Description Use *CreateWnd* to create a Windows control corresponding to the *ScrollingWinControl* object.

CreateWnd initializes the scroll bars on the scrolling windowed control and then calls *TWinControl.CreateWnd*.

See Also *TWinControl.CreateWnd* method

Destroy

Destroy is a method that removes the *TScrollingWinControl* object from memory.

```
destructor Destroy; override;
```

Description Do not call *Destroy*. Instead, call *Free*, which calls *Destroy* if the *TScrollingWinControl* object is not **nil**.

Destroy frees the *HorzScrollBar* and *VertScrollBar* properties and then calls *TWinControl.Destroy*.

See also *TWinControl.Destroy* method

DisableAutoRange

DisableAutoRange disables automatic scrolling.

```
procedure DisableAutoRange;
```

Description Use *DisableAutoRange* to temporarily disable scrolling when rearranging controls on a form. This method can be used in conjunction with the *EnableAutoScroll* method to turn scrolling on and off.

See also *EnableAutoScroll* method

EnableAutoRange

EnableAutoRange enables automatic scrolling.

```
procedure EnableAutoRange;
```

Description Use *EnableAutoRange* to re-enable scrolling after turning it off. This method can be used in conjunction with the *DisableAutoScroll* method to turn scrolling on and off.

See also *DisableAutoScroll* method

ScrollInView

ScrollInView is a method that scrolls a control into the visible area of the scrolling windowed control.

```
procedure ScrollInView(AControl: TControl);
```

Description Use *ScrollInView* to scroll a control into the visible area of the scrolling windowed control.

AControl is a *TControl* object encapsulating the control to scroll into view.

Note If the *ComponentState* property for *AControl* or the scrolling windowed control might be *csLoading*, use *AutoScrollInView* instead.

See also *AutoScrollInView* method, *ComponentState* property, *TControl* object

TServerSocket

Unit: ScktComp — Palette: Internet

TObject • TPersistent • TComponent • TCustomSocket • TCustomServerSocket • TServerSocket

TServerSocket manages server socket connections for a TCP/IP server.

Description Add a *TServerSocket* object to a form or data module to turn an application into a TCP/IP server. *TServerSocket* listens for requests for TCP/IP connections from other machines, and establishes connections when requests are received.

See also *TClientSocket* object, *TServerClientWinSocket* object, *TServerWinSocket* object

S

Properties

The following table lists the properties of *TServerSocket*:

Property	Visibility	Default	Documented in
Active	published		TCustomSocket
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent

Property	Visibility	Default	Documented in
ComponentStyle	public, read-only		TComponent
DesignInfo	public		TComponent
Name	published		TComponent
Owner	public, read-only		TComponent
Port	published		TCustomSocket
ServerType	published		TCustomServerSocket
Service	published		TCustomSocket
Socket	public, read-only		TServerSocket
Tag	published	0	TComponent
ThreadCacheSize	published	10	TCustomServerSocket
VCLComObject	public		TComponent

Socket

Socket is the *TServerWinSocket* object that describes the endpoint of the listening connection.

```
property ServerSocket: TServerWinSocket;
```

Description Use *Socket* to obtain

- Information about the connections currently active on the server socket.
- Information about the connection threads that are cached for use by the server socket.
- Access to the Windows socket handle for the listening connection that is needed for Windows socket API calls.

See also *ThreadCacheSize* property, *TServerWinSocket* object

Events

The following table lists the events of *TServerSocket*:

Event	Visibility	Documented in
OnAccept	published	TCustomSocket
OnClientConnect	published	TCustomServerSocket
OnClientDisconnect	published	TCustomServerSocket
OnClientError	published	TCustomServerSocket
OnClientRead	published	TCustomServerSocket
OnClientWrite	published	TCustomServerSocket
OnGetSocket	published	TCustomServerSocket
OnGetThread	published	TCustomServerSocket
OnListen	published	TCustomSocket
OnThreadEnd	published	TCustomServerSocket
OnThreadStart	published	TCustomServerSocket

Methods

The following table lists the methods of *TServerSocket*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Close	public	TCustomSocket
Create	public	TServerSocket
DefaultHandler	public	TObject
Destroy	public	TCustomServerSocket
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Open	public	TCustomSocket
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Create

Create creates an instance of *TServerSocket*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to create and initialize a server socket component at runtime. *TServerSocket* objects placed at design time are created automatically.

After calling the inherited constructor, *Create*

- Allocates the *TServerWinSocket* object that encapsulates the Windows socket connection for the server socket.
- Initializes the event and error dispatchers for the server socket.
- Initializes *ThreadCacheSize* to 10.

See also *TComponent.Create* method, *ThreadCacheSize* property

TSession

Unit: DBTables | Palette: Data Access

TObject • TPersistent • TComponent • TSession

TSession provides global management of a group of database connections within an application.

Description Use *TSession* to manage a group of database connections within an application. There are three uses for *TSession*: standard, multiple net files for Paradox, and multi-threaded database applications.

Delphi automatically creates a globally accessible default *TSession* component named *Session* for all database applications. The default session component handles standard database connections. An application can control the default session by accessing its properties, events, and methods at runtime.

Database applications that must simultaneously access Paradox tables located in different network locations can establish multiple sessions, one for each network location.

Finally, database applications that must establish multiple, concurrent connections to the same database, such as performing two queries against the same data at once, are multi-threaded applications. Multi-threaded applications must create and maintain one additional session component for each simultaneous connection to a single database server.

Note Applications that maintain multiple sessions can manage them through the *TSessionList* component. A default session list component, called *Sessions*—note the plural which distinguishes this component from the default session component, *Session*—is automatically created for all database applications.

See also *TDatabase* object, *TSessionList* object

Properties

The following table lists the properties of *TSession*:

Property	Visibility	Default	Documented in
Active	published	False	TSession
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ConfigMode	public		TSession
DatabaseCount	public, read-only		TSession
Databases	public, read-only		TSession
DesignInfo	public		TComponent
Handle	public, read-only		TSession
KeepConnections	published	True	TSession
Locale	public, read-only		TSession
Name	published		TComponent
NetFileDir	published		TSession
Owner	public, read-only		TComponent
PrivateDir	published		TSession
SessionName	published		TSession
SQLHourGlass	published	True	TSession
Tag	published	0	TComponent
TraceFlags	public		TSession
VCLComObject	public		TComponent

S

Active

Specifies whether or not a session is active.

property Active: Boolean;

Description Examine *Active* at runtime for the default session, *Session*, to determine if it is active or not. *Active* is *False* by default, meaning there are no open database connections or datasets associated with the session. If *Active* is *True*, the session is active even if there are currently no active database connections.

Set the *Active* property at design time or runtime for additional session components in a database application. Setting *Active* to *True* starts the session and makes it the current session. Setting *Active* to *True* triggers an *OnStartup*

event handler for the session and initializes the *NetFileDir*, *PrivateDir*, and *ConfigMode* properties.

Setting *Active* to *False* (the default), closes any open datasets, and disconnects from attached database servers.

Note An application can close the default session by setting *Active* to *False*, but this is not recommended.

See also *ConfigMode* property, *NetFileDir* property, *PrivateDir* property, *Close* method, *Open* method

AutoSessionName

Indicates whether or not a unique session name is automatically generated for a session.

```
property AutoSessionName: Boolean;
```

Description Set *AutoSessionName* to control whether or not a unique session name is automatically generated by Delphi. *AutoSessionName* is intended to guarantee developers of Web-based applications that sessions spawned for Web servers are unique at runtime.

When *AutoSessionName* is *False* (the default), the application must set the *SessionName* property for a session component. When *AutoSessionName* is *True*, Delphi assigns the *SessionName* property automatically and replicates the session name across the *SessionName* properties of all dataset components in the data module or form where the session component appears. Delphi constructs a session name by taking the current value of the *Name* property and appending an underscore (_) followed by a numeric value. For example, if *Name* is *CustomerSession*, then the *AutoSessionName* might be *CustomerSession_2*.

Note The following restrictions apply to *AutoSessionName*:

- You cannot set *AutoSessionName* for a session in a data module or form that contains more than one session component.
- You cannot add a session component to a data module or form that already contains a session component with *AutoSessionName* set to *True*.
- You cannot directly set the *SessionName* property of a session when *AutoSessionName* is *True*.

See also *Name* property, *SessionName* property

ConfigMode

Specifies how the Borland Database Engine (BDE) should handle aliases for the session.

```
type
  TConfigModes = (cfmVirtual, cfmPersistent, cfmSession);
```

```
TConfigMode = set of TConfigModes;
property ConfigMode: TConfigMode;
```

Description Set *ConfigMode* to specify how the BDE should handle aliases created using the *AddAlias* or *AddStandardAlias* methods within the context of a session. *ConfigMode* also determines which BDE aliases are reported to a session when an application calls the *GetAliasNames* method.

The default setting for *ConfigMode* is *cmAll*, which is a constant that includes all possible set items defined in *TConfigModes*. The following table summarizes the possible BDE configuration modes that can be set for aliases:

Mode	Meaning
cfmVirtual	All aliases in the BDE configuration file, the BDE persistent store, and aliases local to the session are available to the session.
cfmPersistent	Only aliases in the BDE configuration file, or that are added to the persistent store are available to the session.
cfmSession	Only aliases created within this session are available to the session.

The main purpose of *ConfigMode* is to enable an application to specify and restrict aliases at the session level.

Note Aliases created within a session are not automatically saved to the BDE configuration file. To save an alias to the configuration file, call the *SaveConfigFile* method.

See also *AddAlias* method, *AddStandardAlias* method, *GetAliasNames* method, *SaveConfigFile* method

DatabaseCount

Indicates the number of active database components currently associated with the session.

```
property DatabaseCount: Integer;
```

Description Examine *DatabaseCount* to determine the number of active database components associated with a session. As database connections are opened or closed during a session's life-span, this number can change. If *DatabaseCount* is zero, there are currently no active database components for the session.

DatabaseCount is typically used with the *Databases* property to iterate through the current set of active databases in a session. For example, the following code sets the *KeepConnection* property of each active database in the default session to *True*:

```
var
  MaxDbCount: Integer;
begin
  with Session do
    if DatabaseCount > 0 then
```

```
    begin
      for MaxDbCount := 1 to DatabaseCount do
        Databases[MaxDbCount].KeepConnection : = True;
    end;
  end;
end;
```

See also *Databases* property, *TDatabase* object

Databases

Provides an indexed array of all active database component names for a session.

property Databases[Index: Integer]: TDatabase;

Description Use *Databases* to access active database components associated with a session. An active database is one that is currently connected to a database server.

Databases is typically used with the *DatabaseCount* property to iterate through the current set of active databases in a session. For example, the following code sets the *KeepConnection* property of each active database in the default session to *True*:

```
var
  MaxDbCount: Integer;
begin
  with Session do
    if DatabaseCount > 0 then
    begin
      for MaxDbCount := 1 to DatabaseCount do
        Databases[MaxDbCount].KeepConnection : = True;
    end;
  end;
end;
```

See also *DatabaseCount* property, *TDatabase* object

Handle

Specifies the Borland Database Engine (BDE) handle for the session.

type HDBISES: Longint;

property Handle: HDBISES;

Description Use *Handle* only to bypass *TSession* methods and write directly to the BDE API. Many BDE function calls require a handle parameter. *Handle* is assigned an initial value when a session is activated.

Note Do not use this property unless an application requires BDE functionality not available through standard Delphi components.

See also *Locale* property

KeepConnections

Specifies whether or not a temporary database component created in the context of a session maintains a database server connection even if there are no active datasets associated with the database component.

```
property KeepConnections: Boolean;
```

Description Use *KeepConnections* to specify whether or not temporary database components created at runtime maintain server connections even when they have no active datasets. If *KeepConnections* is *True* (the default), the application maintains database connections until the application exits or calls the *DropConnections* method. For connections to remote database servers, or for applications that frequently open and close datasets, *KeepConnections* should remain *True* to reduce network traffic and speed up processing.

When *KeepConnections* is *False*, an application disconnects from a database when all datasets associated with a database component are closed. Dropping a connection releases system resources allocated to the connection, but if a dataset is later reopened that uses the same database, the connection must be reestablished and initialized.

Note Duration of server connection for a persistent database component is determined by the database component's *KeepConnection* property instead of the session's *KeepConnections* property.

See also *DropConnections* method, *TDatabase* object, *TDatabase.KeepConnection* property

Locale

Identifies the Borland Database Engine (BDE) language driver for the session component.

```
type TLocale: Pointer;
```

```
property Locale: TLocale;
```

Description Examine *Locale* to determine the BDE language driver used by the session component. Applications that make direct calls to the BDE may need to pass *Locale* information as an API function parameter.

Note Do not use this property unless an application requires BDE functionality not available through standard Delphi components.

See also *Handle* property

NetFileDir

Specifies the directory that contains the Borland Database Engine (BDE) network control file, PDOXUSRS.NET.

```
property NetFileDir: string;
```

Description Use *NetFileDir* to specify the directory that contains the BDE network control file, PDOXUSER.NET. This file governs the sharing of Paradox tables on network drives. All applications that share Paradox tables on a network directory must specify the same value for this property, and must have read, write, and create rights for the directory.

Delphi derives an initial value for *NetFileDir* from the BDE configuration file for the Paradox driver. Any value assigned to *NetFileDir* at design time or runtime overrides the BDE configuration setting.

See also *PrivateDir* property

PrivateDir

Specifies the directory in which to store temporary table processing files generated by the Borland Database Engine (BDE) for database components associated with a session.

```
property PrivateDir: string;
```

Description Use *PrivateDir* to set the directory in which to store temporary table processing files for all database connections, such as those generated by the BDE to handle local SQL statements. Ordinarily this value is only set at runtime, so that a user's local hard disk is used to store temporary files. Local storage of these files improves performance. If no value is specified for *PrivateDir*, the BDE automatically stores temporary files in the directory current at the time the BDE was initialized.

Note For applications that run directly from a networked file server, the application should set *PrivateDir* to a user's local drive to improve performance and to prevent temporary files from being created on the server where they might conflict with temporary files created by other instances of the application.

See also *NetFileDir* property

SessionName

Specifies a unique session name that can be used by database and dataset components to link to this session.

```
property SessionName: string;
```

Description Set *SessionName* to a unique and meaningful name that can be used to link database and dataset components to the session. Database and dataset components each have corresponding *SessionName* properties that must either match the *SessionName* property of an active session or that be blank, indicating that they should be associated with the default session, *Session*.

Note If the *AutoSessionName* property is *True*, an application cannot set *SessionName* directly.

See also *TDatabase* object, *TDataSet* object, *TDatabase.SessionName* property, *TDataSet.SessionName* property

SQLHourGlass

Indicates whether or not the mouse cursor changes to an hourglass during BDE operations.

```
property SQLHourGlass: Boolean;
```

Description Set *SQLHourGlass* to *False* if you do not want the mouse cursor to change to an hourglass while the BDE is waiting for an SQL server to perform an operation. This property must be set before the session is opened.

TraceFlags

Specifies the database operations to track with the SQL Monitor at runtime.

```
type
  TTraceFlag = (tfQPrepare, tfQExecute, tfError, tfStmt, tfConnect, tfTransact,
  tfBlob, tfMisc, tfVendor, tfDataIn, tfDataOut);

  TTraceFlags = set of TTraceFlag;

property TraceFlags: TTraceFlags;
```

Description Use *TraceFlags* to specify which database operations the SQL Monitor should track in an application at runtime. *TraceFlags* is only meaningful for the SQL Monitor, which is provided to enable performance tuning and SQL debugging when working with remote SQL database servers.

Note Normally trace options are set from the SQL Monitor rather than setting *TraceFlags* in application code.

The value of a session component's *TraceFlags* property determines the initial settings of the *TraceFlags* property for database components associated with the session.

The *TTraceFlags* type defines the individual values that can be included in the *TraceFlags* property. The following table summarizes those values:

Values	Meaning
tfQPrepare	Monitor Prepare statements.
tfQExecute	Monitor ExecSQL statements.
tfError	Monitor server error messages. Such messages may include an error code.
tfStmt	Monitor all SQL statements.
tfConnect	Monitor database connect and disconnect operations, including allocation of connection handles, and freeing connection handles.
tfTransact	Monitor transaction statements, such as StartTransaction, Commit, and Rollback.
tfBlob	Monitor operations on blob data types.

Values	Meaning
tfMisc	Monitor any statements not covered by other flag options.
tfVendor	Monitor direct API function calls to the database server.
tfDataIn	Monitor data received from a server.
tfDataOut	Monitor data sent to a server.

Because *TraceFlags* is a set property, an application can specify different combinations of flags to monitor different combinations of statements. For example, the following statement limits monitoring to database connections and SQL statement preparation:

```
TraceFlags := [tfConnect, tfQPrepare];
```

See also *SessionName* property, *TDatabase.TraceFlags* property

Events

The following table lists the events of *TSession*:

Event	Visibility	Documented in
OnPassword	published	TSession
OnStartup	published	TSession

OnPassword

OnPassword occurs when an application attempts to open a Paradox table for the first time and the Borland Database Engine (BDE) reports insufficient access rights.

```
type TPasswordEvent = procedure(Sender: TObject; var Continue: Boolean) of object;

property OnPassword: TPasswordEvent;
```

Description Write an *OnPassword* event handler to take specific action when an application attempts to open a password-protected Paradox table for the first time. To gain access to the Paradox table, the event handler must pass a valid password to the BDE. The event handler should call the *AddPassword* method to make the password available to the session.

Note If an *OnPassword* event does not exist, but the BDE reports insufficient access rights, Delphi displays a default dialog box that prompts for a password.

See also *NetFileDir* property, *PrivateDir* property, *AddPassword* method, *RemovePassword* method

OnStartup

OnStartup occurs when an application activates a session.

```
type TNotifyEvent = procedure(Sender: TObject) of object;
property OnStartup: TNotifyEvent;
```

Description Write an *OnStartup* event handler to take specific actions when an application activates a session. A session is activated by setting its *Active* property to *True*, or by opening a database or dataset associated with the session (both of which set *Active* to *True*).

See also *Active* property, *Open* method, *OpenDatabase* method

Methods

The following table lists the methods of *TSession*:

Method	Visibility	Documented in
AddAlias	public	TSession
AddDriver	public	TSession
AddPassword	public	TSession
AddStandardAlias	public	TSession
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Close	public	TSession
CloseDatabase	public	TSession
Create	public	TSession
DefaultHandler	public	TObject
DeleteAlias	public	TSession
DeleteDriver	public	TSession
Destroy	public	TSession
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
DropConnections	public	TSession
FieldAddress	public	TObject
FindComponent	public	TComponent
FindDatabase	public	TSession
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent

Method	Visibility	Documented in
GetAliasDriverName	public	TSession
GetAliasNames	public	TSession
GetAliasParams	public	TSession
GetConfigParams	public	TSession
GetDatabaseNames	public	TSession
GetDriverNames	public	TSession
GetDriverParams	public	TSession
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetPassword	public	TSession
GetStoredProcNames	public	TSession
GetTableNames	public	TSession
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
IsAlias	public	TSession
MethodAddress	public	TObject
MethodName	public	TObject
ModifyAlias	public	TSession
ModifyDriver	public	TSession
NewInstance	public	TObject
Open	public	TSession
OpenDatabase	public	TSession
RemoveAllPasswords	public	TSession
RemoveComponent	public	TComponent
RemovePassword	public	TSession
SafeCallException	public	TComponent
SaveConfigFile	public	TSession

AddAlias

Adds a specified Borland Database Engine (BDE) alias to the session for an SQL database server.

```
procedure AddAlias(const Name, Driver: string; List: TStrings);
```

Description Call *AddAlias* at runtime to create a session-specific BDE alias for an SQL database server. *Name* is the unique name to assign to the alias. *Driver* is the

name of the SQL Links database driver to use for the alias. *List* is a string list containing parameters for the alias.

A session's *ConfigMode* property determines whether an alias is local to the session, or is available to all applications in the BDE persistent store as long as any BDE clients are active. the lifetime of the session. To save an alias permanently, call the *SaveConfigFile* method before ending the session.

Note Although *AddAlias* can be used to create a Paradox or dBASE alias, *AddStandardAlias* provides a simpler method for Paradox and dBASE alias creation.

Example This example creates an alias for an InterBase database server:

```
var
  MyList: TStringList;
begin
  MyList := TStringList.Create;
  try
    with MyList do
    begin
      Add('SERVER NAME=IB_SERVER:/PATH/DATABASE.GDB');
      Add('USER NAME=MYNAME');
    end;
    Session1.AddAlias('NewIBAlias', 'INTRBASE', MyList);
  finally
    MyList.Free;
  end;
end;
```

See also *AddStandardAlias* method, *ConfigMode* property, *DeleteAlias* method, *IsAlias* method, *ModifyAlias* method, *SaveConfigFile* method

AddDriver

Adds a specified Borland Database Engine (BDE) driver to the session for an SQL database server.

procedure AddDriver(**const** Name: **string**; List: TStrings);

Description Call *AddDriver* at runtime to create a BDE driver for an SQL database server. *AddDriver* can also be used to create ODBC driver entries. *Name* is the unique name to assign to the driver. *List* is a string list containing parameters for the alias.

A session's *ConfigMode* property determines whether a driver is local to the session, or is available to all applications in the BDE persistent store as long as any BDE clients are active. To save a driver permanently, call the *SaveConfigFile* method before ending the session.

See also *ConfigMode* property, *DeleteDriver* method, *IsAlias* method, *SaveConfigFile* method

AddPassword

Adds a password to the current session for accessing encrypted Paradox tables.

```
procedure AddPassword(const Password: string);
```

Description Call *AddPassword* to provide a password for a session prior to opening an encrypted Paradox table that requires a password. If an application opens a Paradox table that requires a password for access, the session does not have a password assigned to it, and the application does not provide its own *OnPassword* event handler, Delphi displays a dialog box prompting the user for a valid password before allowing table access.

Note If an application defines its own *OnPassword* event handler, the handler should call this method to add passwords for the session. All passwords assigned by *AddPassword* should be deleted with *RemoveAllPasswords* or *RemovePassword* before an application closes.

See also *OnPassword* event, *RemoveAllPasswords* method, *RemovePassword* method

AddStandardAlias

Adds a standard Borland Database Engine (BDE) alias to the session for Paradox, dBASE, or ASCII tables.

```
procedure AddStandardAlias(const Name, Path, DefaultDriver: string);
```

Description Call *AddStandardAlias* at runtime to create a session-specific BDE alias for Paradox, dBASE or ASCII tables. *Name* is the unique name to assign to the alias. Path specifies the directory where the database is located. *DefaultDriver* is the name of the standard BDE driver to use for the alias. If set, *DefaultDriver* must be one of "Paradox," "DBASE," or "ASCIIDRV." If left blank *DefaultDriver* defaults to "Paradox." *DefaultDriver* is used when attempting to open a table that both does not have an extension and is defined as a table of type *ttDefault*.

A session's *ConfigMode* property determines whether an alias is local to the session, or is available to all applications in the BDE persistent store as long as any BDE clients are active. the lifetime of the session. To save an alias permanently, call the *SaveConfigFile* method before ending the session.

Note To create an alias for an SQL database, call *AddAlias* instead of *AddStandardAlias*.

See also *AddAlias* method, *ConfigMode* property, *DeleteAlias* method, *IsAlias* method, *ModifyAlias* method, *SaveConfigFile* method

Close

Disconnects from all databases, and closes the session.

```
procedure Close;
```

Description Call *Close* to end a session. *Close* disconnects from all active databases by freeing temporary database components, and calling the *Close* methods persistent database components, and then sets its own session handle to **nil**.

Note Setting the *Active* property to *False* also closes a session.

See also *Active* property, *Open* method, *TDatabase.Close* method

CloseDatabase

Closes a database connection associated with the current session.

```
procedure CloseDatabase(Database: TDatabase);
```

Description Call *CloseDatabase* to close a persistent database connection, or to close and free a temporary database connection. *Database* specifies the name of the database component for which to close the connection.

CloseDatabase decrements the specified database component's reference count, and then, if the reference count is zero and the database component's *KeepConnection* property is *False*, *CloseDatabase* either frees a temporary database component or closes the connection for a persistent database component.

Note Calling *CloseDatabase* for a persistent database component does not close the connection. To close a connection for a persistent database component, call the database component's *Close* method directly.

Temporary database components are closed automatically when the last dataset associated with the database component is closed, but an application can call *CloseDatabase* prior to that time to force closure. Closing a connection established by a temporary database component does not free the component if the session's *KeepConnections* property is *True* (the default). To free temporary database components after closing their connections call *DropConnections*.

See also *DropConnections* method, *KeepConnections* property, *OpenDatabase* method, *TDatabase.Close* method, *TDatabase.KeepConnection* property

S

Create

Creates an instance of a *TSession* component.

```
constructor Create(AOwner: TComponent);
```

Description Do not call *Create* directly. The default session component, *Session*, is created automatically for all database applications. To create additional sessions for a multi-threaded database application at runtime call *Sessions.OpenSession* instead of *Create*. *OpenSession* calls *Create* only if the session does not already exist. (At design time, create additional sessions by dropping them on a data module.)

Create calls its inherited constructor and ensures that the session component cannot be inherited should the data module or form on which it is placed be stored in the Object Repository. *Create* also:

- Creates an empty list of database components for the session.
- Sets the *KeepConnections* property to *True*.
- Adds the session to the session list.
- Sets the *Handle* property to **nil**.

See also *Destroy* method, *Handle* property, *KeepConnections* property, *TSessionList.OpenSession* method

DeleteAlias

Removes a specified Borland Database Engine (BDE) alias from the session.

```
procedure DeleteAlias(const Name: string);
```

Description Call *DeleteAlias* to remove a BDE alias from a session. Removing an alias makes the alias unavailable to the session, but does not remove it from the BDE configuration file if the alias was stored there when it was created. To remove an alias from the BDE configuration file follow the call to *DeleteAlias* with a call to *SaveConfigFile*.

See also *AddAlias* method, *AddStandardAlias* method, *ConfigMode* property, *IsAlias* method, *SaveConfigFile* method

DeleteDriver

Removes a specified Borland Database Engine (BDE) driver from the session.

```
procedure DeleteDriver(const Name: string);
```

Description Call *DeleteDriver* to remove a BDE driver from a session. Removing a driver makes the driver unavailable to the session, but does not remove it from the BDE configuration file if the driver was stored there when it was created. To remove a driver from the BDE configuration file follow the call to *DeleteDriver* with a call to *SaveConfigFile*.

See also *AddDriver* method, *ConfigMode* property, *SaveConfigFile* method

Destroy

Destroys the instance of a session component.

```
destructor Destroy; override;
```

Description Do not call *Destroy* directly. For session components other than the default session, an application should call *Free*, which verifies that the session component is not already freed before calling *Destroy*.

Warning Never call *Free* or *Destroy* for the default session. Freeing of the default session is handled automatically by Delphi when an application terminates.

Destroy performs the following actions before calling its inherited destructor:

- Makes the session inactive, which closes all database connections.
- Removes the session from the session list.
- Resets database session references, if necessary.
- Frees the list of database components associated with the session.

See also *Active* property, *Create* method, *OpenSession* method, *TObject.Free* method, *TSessionList* object

DropConnections

Frees all temporary database components associated with a session if they are inactive.

```
procedure DropConnections;
```

Description Use *DropConnections* to free all temporary database components that are currently inactive. If the *KeepConnections* property of the session is *True* (the default), then temporary database components created as needed for the session by Delphi at runtime are not automatically freed when their database connections are closed. *DropConnections* enables an application to free these components when they are no longer needed.

See also *CloseDatabase* method, *KeepConnections* property

FindDatabase

Searches a session's list of database components for a specified database.

```
function FindDatabase(const DatabaseName: string): TDatabase;
```

Description Call *FindDatabase* to determine whether or not a specified database is associated with a session. *DatabaseName* specifies the name of the database component for which to search.

FindDatabase compares *DatabaseName* to the *DatabaseName* property for each database component in a session's list. If a match is found, *FindDatabase* returns a pointer to the database component. Otherwise *FindDatabase* returns **nil**.

An application can call *FindDatabase* to see if a database component already exists before creating a new database component for a session.

See also *Databases* property, *OpenDatabase* method, *TDatabase.DatabaseName* property

GetAliasDriverName

Provides the name of the database driver used by a specified Borland Database Engine (BDE) alias associated with the session.

```
function GetAliasDriverName(const AliasName: string): string;
```

Description Call *GetAliasDriverName* to determine the BDE database driver used by a specific alias. *AliasName* is the name of the alias for which to return driver information.

For aliases corresponding to Paradox and dBASE tables, *GetAliasDriverName* returns "STANDARD." For ASCII tables the return value is "ASCIIDRV." For SQL databases, the driver name depends on the database server.

See also *GetAliasNames* method, *GetAliasParams* method, *GetDriverNames* method, *GetDriverParams* method

GetAliasNames

Populates a string list with the names of persistent Borland Database Engine (BDE) aliases.

```
procedure GetAliasNames(List: TStrings);
```

Description Call *GetAliasNames* to retrieve a list of persistent BDE aliases—those aliases that exist in persistent storage or in the BDE configuration file. *List* is a string list object, created and maintained by the application, into which to store the alias names.

A session's *ConfigMode* property determines whether an alias created in the context of this session is reported by *GetAliasNames*. If *ConfigMode* is *cmAll*, aliases created within the session are reported.

See also *AddAliasName* method, *AddStandardAlias* method, *ConfigMode* property, *GetAliasDriverName* method, *GetAliasParams* method, *SaveConfigFile* method

GetAliasParams

Retrieves the parameters associated with a specified Borland Database Engine (BDE) alias.

```
procedure GetAliasParams(const AliasName: string; List: TStrings);
```

Description Call *GetAliasParams* to retrieve the BDE configuration parameters for a specified alias. *AliasName* is the name of the alias for which to retrieve parameters. *List* is a string list object, created and maintained by the application, into which to store the parameters.

See also *GetAliasDriverName* method, *GetConfigParams* method

GetConfigParams

Retrieves Borland Database Engine (BDE) configuration information.

```
procedure GetConfigParams(const Path, Section: string; List: TStrings);
```

Description Call *GetConfigParams* to retrieve specific types of configuration information from the BDE configuration file used to initialize the application.

Path represents the internal node structure of the BDE configuration file, and must always start at the root, denoted by a backslash. Each node must be separated from others by a backslash, and no node may be empty. For example, a *Path* of \SYSTEM\FORMATS\TIME, returns the time format used by the BDE configuration file.

Path can contain a replaceable string formatting symbol (%s) in one of its node designations. If it does, *Section* must contain the missing node information. In this case, *GetConfigParams* inserts the contents of *Section* into *Path* before calling the BDE for the requested information.

List is a string list object, created and maintained by the application, into which to store the configuration information. For example, if *Path* is \DATABASES\%s\DB INFO, and *Section* is an alias name, then *GetConfigParams* returns the parameters for the alias in *List*.

For more information about the BDE Configuration File, see the online BDE help.

See also *GetAliasDriverName* method, *GetAliasNames* method, *GetAliasParams* method, *GetDatabaseNames* method, *GetDriverNames* method, *GetDriverParams* method, *GetStoredProcNames* method, *GetTableNames* method

GetDatabaseNames

Populates a string list with the names of persistent Borland Database Engine (BDE) aliases and the names of all databases components known to the session.

```
procedure GetDatabaseNames(List: TStrings);
```

Description Call *GetDatabaseNames* to retrieve a sorted, combined list of persistent BDE aliases—those aliases that exist in persistent storage or in the BDE configuration file—and the names of all database components known to the session. *List* is a string list object, created and maintained by the application, into which to store the database and alias names.

See also *GetAliasNames* method, *GetDriverNames* method, *SaveConfigFile* method

GetDriverNames

Populates a string list with the names of all Borland Database Engine (BDE) drivers available to the session.

```
procedure GetDriverNames(List: TStrings);
```

Description Call *GetDriverNames* to retrieve the names of all BDE drivers that can be used by a session. *List* is a string list object, created and maintained by the application, into which to store the driver names.

See also *GetDriverParams* method

GetDriverParams

Populates a string list with the parameters for a specified Borland Database Engine (BDE) driver.

```
procedure GetDriverParams(const DriverName: string; List: TStrings);
```

Description Call *GetDriverParams* to retrieve a list of parameters associated with a specified BDE driver. *DriverName* is the BDE driver for which to return a parameter list. *List* is a string list object, created and maintained by the application, which stores the driver parameters.

The actual number of parameters returned by *GetDriverParams* depends on the specified driver. For example, for the STANDARD driver used by Paradox and dBASE, returns only a few parameters. For SQL Links drivers multiple parameters are returned.

See also *GetDriverNames* method

GetPassword

Invokes the *OnPassword* event handler for the session or displays the default password dialog box.

```
function GetPassword: Boolean;
```

Description Call *GetPassword* to invoke the *OnPassword* event handler for the session and handle response logic when an application is working with password-protected Paradox files. If an *OnPassword* event handler is not defined, *GetPassword* displays the default password dialog box.

An application can use the return value of *GetPassword* to control program logic. *GetPassword* returns *True* if a user chooses OK, or *False* if the user chooses Cancel.

See also *AddPassword* method, *OnPassword* event, *RemoveAllPasswords* method, *RemovePassword* method

GetStoredProcNames

Populates a string list with the names of all stored procedures associated with a specified database component that is connected to an SQL database server.

```
procedure GetStoredProcNames(const DatabaseName: string; List: TStrings);
```

Description Call *GetStoredProcNames* to retrieve a list of all stored procedures for a database component that is attached to an SQL database server. *DatabaseName* specifies the name of the database for which to retrieve stored procedure names. *List* is a string list object, created and maintained by the application, into which to return the stored procedure names.

Note *GetStoredProcNames* raises an exception if called on Paradox or dBASE databases.

See also *GetAliasDriverName* method, *GetAliasNames* method, *GetDriverNames* method, *GetTableNames* method

GetTableNames

Populates a string list with the names of tables associated with a specified database component.

```
procedure GetTableNames(const DatabaseName, Pattern: string; Extensions,
  SystemTables: Boolean; List: TStrings);
```

Description Call *GetTableNames* to retrieve a list of the tables associated with a given database. *DatabaseName* specifies the name of the database from which to retrieve table names.

Pattern specifies a delimiter string that restricts the tables returned to those that match the string. *Pattern* can include wildcard symbols. Pass an empty *Pattern* string to match all files not restricted by other criteria.

For searches on Paradox and dBASE, set *Extensions* to *True* to include filename extensions as part of the table name. For SQL-based databases, set *Extension* to *False*.

Set *SystemTables* to *True* for SQL-based databases to return both data tables and the system tables that define the database structure. Set *SystemTables* to *False* for Paradox and dBASE tables.

List is a string list object, created and maintained by the application, into which to return the table names.

See also *GetAliasDriverName* method, *GetAliasNames* method, *GetDatabaseNames* method, *GetDriverNames* method, *GetStoredProcNames* method

S

IsAlias

Determines whether or not a string value corresponds to an existing Borland Database Engine (BDE) database alias known to the session.

```
function IsAlias(const Name: string): Boolean;
```

Description Call *IsAlias* to determine if a string corresponds to the name of an existing BDE alias that can be used by the session. *Name* is the string to test.

If *IsAlias* returns *True*, *Name* is a valid alias.

Note The aliases known to *IsAlias* depend on the session's *ConfigMode* property. If *ConfigMode* is *cmAll*, then all aliases are available to the session.

See also *AddAlias* method, *AddStandardAlias* method, *ConfigMode* property, *DeleteAlias* method, *GetAliasNames* method, *GetAliasParams* method, *ModifyAlias* method

ModifyAlias

Adds or changes parameters for a Borland Database Engine (BDE) alias.

```
procedure ModifyAlias(Name: string; List: TStrings);
```

Description Call *ModifyAlias* to change the parameters for a specified BDE alias. *Name* is the name of the alias to modify. *List* is a string list object previously populated with parameter values to add or change. An application must create and maintain *List*.

Populate *List* with parameter strings before calling *ModifyAlias*. *List* need only contain the parameters to change. If *ModifyAlias* passes a parameter for which a value already exists, the new parameter replaces the existing one. New parameters that are not yet defined for the alias are added to the existing parameters. If *List* is empty, *ModifyAlias* does not change any existing parameters.

See also *ConfigMode* property, *GetAliasParams* property, *TDatabase.Params* property

Open

Starts a session and makes it the current session.

```
procedure Open;
```

Description Call *Open* to start a session other than the default session in a database application. *Open* sets *Active* to *True*, triggers an *OnStartup* event handler for the session and initializes the *NetFileDir*, *PrivateDir*, and *ConfigMode* properties.

See also *Active* property, *Close* method, *ConfigMode* property, *NetFileDir* property, *OnStartup* event, *PrivateDir* property

OpenDatabase

Opens an existing database, or creates a temporary database component and opens it.

```
function OpenDatabase(const DatabaseName: string): TDatabase;
```

Description Call *OpenDatabase* to connect to a database for which a persistent database component already exists, or to create a temporary database component and connect to a database. *DatabaseName* specifies the database to open.

OpenDatabase makes the current session active, then calls *FindDatabase* to determine if the *DatabaseName* parameter corresponds to the *DatabaseName*

property of an existing database component. If it does not, *OpenDatabase* creates a temporary database component, assigning the *DatabaseName* parameter to the *DatabaseName* property. Finally, *OpenDatabase* calls the *Open* method of the database to connect to a database server, then increments the session's database reference count by one.

See also *CloseDatabase* method, *FindDatabase* method, *TDatabase.DatabaseName* property, *TDatabase.Open* method

RemoveAllPasswords

Deletes all passwords previously added to the current session for accessing encrypted Paradox tables.

```
procedure RemoveAllPasswords;
```

Description Call *RemoveAllPasswords* to delete all current Paradox table passwords associated with the current session. Subsequent attempts to access encrypted Paradox tables fail unless an application first calls *AddPassword* to reestablish a password.

Note Ordinarily an application should call *RemoveAllPasswords* to remove all session-specific passwords it creates from the Borland Database Engine (BDE) persistent store before terminating.

See also *AddPassword* method, *RemovePassword* method

RemovePassword

Deletes a single password previously added to the current session for accessing an encrypted Paradox table.

```
procedure RemovePassword(const Password: string);
```

Description Call *RemovePassword* to delete a single Paradox table password associated with the current session from the Borland Database Engine (BDE) persistent store. Subsequent attempts to access the encrypted Paradox table fails unless an application first calls *AddPassword* to reestablish a password.

See also *AddPassword* method, *RemoveAllPasswords* method

SaveConfigFile

Writes the current information in the Borland Database Engine (BDE) persistent store from memory to the BDE configuration file on disk.

```
procedure SaveConfigFile;
```

Description Call *SaveConfigFile* to write any BDE configuration changes currently in memory to the BDE configuration file on disk. When the BDE is first initialized, the current BDE configuration file is read into persistent storage in memory. Applications can read or modify the configuration file in memory, but modification exist only in memory, not on disk.

For example, if an application defines a new BDE alias with *AddAlias*, the alias exists in memory only for as long as the BDE resides in memory. To store the new alias for later use by all applications that use the BDE, call *SaveConfigFile*. *SaveConfigFile* overwrites the disk image of the configuration file with the in-memory image of the persistent configuration file.

Note If a session's *ConfigMode* property is *cmSession*, added aliases cannot be saved to the BDE configuration file.

See also *AddAlias* method, *ConfigMode* property

TShape

Unit: ExtCtrls | Palette: Additional

TObject • TPersistent • TComponent • TControl • TGraphicControl • TShape

TShape represents a geometric shape that can be drawn on a form.

Description Add a *TShape* object to a form to draw a simple geometric shape on the form. *TShape* introduces properties to describe the pen used to outline the shape and the brush used to fill it.

If the shape is only part of the image of a *TCustomControl* object, use the methods of the control's canvas instead.

See also *TBrush* object, *TCanvas* object, *TPen* object

Properties

The following table lists the properties of *TShape*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
BoundsRect	public		TControl
Brush	published		TShape
ClientHeight	public		TControl
ClientOrigin	public, read-only		TControl
ClientRect	public, read-only		TControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlState	public		TControl

Property	Visibility	Default	Documented in
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Height	published		TControl
Hint	published		TControl
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentShowHint	published	True	TControl
Pen	published		TShape
Shape	published	stRectangle	TShape
ShowHint	published		TControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Brush

Brush specifies the color and pattern used for filling the shape control.

property Brush: TBrush;

Description Read *Brush* to obtain the *TBrush* object that represents the color and pattern that fill the interior of the shape. Use the properties of *Brush* to change the color or pattern.

Set the *Brush* property to assign another *TBrush* object to the *TBrush* object maintained by *TShape*.

See also *TBrush* object

Pen

Pen specifies the pen used to outline the shape control.

property Pen: TPen;

Description Read *Pen* to obtain the *TPen* object that represents the color and style of the line used to outline the shape. Use the properties of *Pen* to change the color or style.

Set the *Pen* property to assign another *TPen* object to the *TPen* object maintained by *TShape*.

See also *TPen* object

Shape

Shape specifies the shape of the control.

```
type TShapeType = (stRectangle, stSquare, stRoundRect, stRoundSquare, stEllipse, stCircle);
property Shape: TShapeType;
```

Description Set *Shape* to the geometric shape that should be drawn on the form. The *Shape* property has these possible values:

Value	Meaning
stCircle	The shape is a circle.
stEllipse	The shape is an ellipse.
stRectangle	The shape is a rectangle.
stRoundRect	The shape is a rectangle with rounded corners.
stRoundSquare	The shape is a square with rounded corners.
stSquare	The shape is a square.

Events

The following table lists the events of *TShape*:

Event	Visibility	Documented in
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TShape*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
Create	public	TShape
DefaultHandler	public	TObject
Destroy	public	TShape
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTextBuf	public	TControl
GetTextLen	public	TControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject

Method	Visibility	Documented in
Invalidate	public	TControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Perform	public	TControl
Refresh	public	TControl
RemoveComponent	public	TComponent
Repaint	public	TControl
SafeCallException	public	TComponent
ScreenToClient	public	TControl
SendToBack	public	TControl
SetBounds	public	TControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TControl

Create

Create creates an instance of *TShape*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to instantiate a *TShape* object at runtime. Shapes added to a form at design time are created automatically.

Create allocates memory and initializes the *Height* and *Width* properties of the shape to 65.

See also *Height* property, *TGraphicControl.Create* method, *Width* property

Destroy

Destroy disposes of an instance of *TShape*.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, an application should call *Free*. *Free* verifies that the object has not already been freed before it calls *Destroy*.

See also *Free* method

TSmallIntField

Unit: DB

TObject • TPersistent • TComponent • TField • TNumericField • TIntegerField • TSmallIntField

Description

A *TSmallintField* object represents a signed 16-bit integer field in a dataset.

Small-integer fields can hold values in the range -32768 to 32767, and are used to hold values that are signed 16-bit whole numbers.

TSmallIntField differs from its immediate ancestor *TIntegerField* only in the data type it represents. Except for the constructor and the *DataSize* property, all of the properties, methods, and events are the same as those of *TIntegerField*, and provide functionality that is useful for managing the value and properties of an integral numeric field in a database.

See also *TDataSet object*, *TWordField object*

Properties

The following table lists the properties of *TSmallIntField*:

Property	Visibility	Default	Documented in
Alignment	published	taRightJustify	TField
AsBoolean	public		TField
AsCurrency	public		TField
AsDateTime	public		TField
AsFloat	public		TIntegerField
AsInteger	public		TIntegerField
AsString	public		TIntegerField
AsVariant	public		TIntegerField
AttributeSet	public		TField
Calculated	public	False	TField
CanModify	public, read-only		TField
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ConstraintErrorMessage	published		TField
CurValue	public, read-only		TField
CustomConstraint	published		TField
DataSet	public		TField
DataSize	public, read-only		TSmallintField

S

Property	Visibility	Default	Documented in
DataType	public, read-only		TField
DefaultExpression	published		TField
DesignInfo	public		TComponent
DisplayFormat	published		TNumericField
DisplayLabel	published		TField
DisplayName	public, read-only		TField
DisplayText	public, read-only		TField
DisplayWidth	published		TField
EditFormat	published		TNumericField
EditMask	public		TField
EditMaskPtr	public, read-only		TField
FieldKind	published	fkData	TField
FieldName	published		TField
FieldNo	public, read-only		TField
ImportedConstraint	published		TField
Index	published		TField
IsIndexField	public, read-only		TField
IsNull	public, read-only		TField
KeyFields	published		TField
Lookup	public		TField
LookupCache	published	False	TField
LookupDataSet	published		TField
LookupKeyFields	published		TField
LookupList	public, read-only		TField
LookupResultField	published		TField
MaxValue	published	0	TIntegerField
MinValue	published	0	TIntegerField
Name	published		TComponent
NewValue	public		TField
Offset	public, read-only		TField
OldValue	public, read-only		TField
Origin	published		TField
Owner	public, read-only		TComponent
ReadOnly	published	False	TField
Required	published	False	TField
Size	public		TField
Tag	published	0	TComponent
Text	public		TField
ValidChars	public		TField
Value	public		TIntegerField
VCLComObject	public		TComponent
Visible	published	True	TField

DataSize

DataSize is the amount of memory that is needed to store the value of a small integer field in its native database format.

```
property DataSize: Word;
```

Description Check *DataSize* to determine the number of bytes required to store a small integer field component's value. Use *DataSize* to determine the size needed for a buffer when working with the *GetData* and *SetData* methods.

For small integer fields, *DataSize* is always 2. This is the size of 16-bit integer values in the database tables, not the size of the LongInt data type used to store and manipulate integer values by *TSmallIntField*.

See also *GetData* method, *SetData* method, *Value* property

Events

The following table lists the events of *TSmallIntField*:

Event	Visibility	Documented in
OnChange	published	TField
OnGetText	published	TField
OnSetText	published	TField
OnValidate	published	TField

Methods

The following table lists the methods of *TSmallIntField*:

Method	Visibility	Documented in
Assign	public	TField
AssignValue	public	TField
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TField
Create	public	TSmallintField
DefaultHandler	public	TObject
Destroy	public	TField
DestroyComponents	public	TComponent
Destroying	public	TComponent

S

Method	Visibility	Documented in
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
FocusControl	public	TField
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetData	public	TField
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
IsBlob	public	TField
IsValidChar	public	TField
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RefreshLookupList	public	TField
RemoveComponent	public	TComponent
SafeCallException	public	TComponent
SetData	public	TField
SetFieldType	public	TField
Validate	public	TField

Create

Create creates an instance of a *TSmallintField* object.

constructor Create(AOwner: TComponent); **override**;

Description It is seldom necessary to call *Create* directly, because a small-integer field object is instantiated automatically for all small-integer fields in a dataset.

After calling the inherited constructor, *Create* sets

- *DataType* to *ftSmallint*
- *MinValue* to –32768
- *MaxValue* to 32767

See also *DataType* property, *MaxValue* property, *MinValue* property, *TField.Create* method

TSpeedButton

Unit: Buttons | Palette: Additional

TObject • TPersistent • TComponent • TControl • TGraphicControl • TSpeedButton

TSpeedButton is a button that is used to execute commands or set modes.

Description Use *TSpeedButton* to add a button to a group of buttons in a form. *TSpeedButton* introduces properties that can be used to set graphical images that represent the different button states (selected, unselected, disabled and so on). Use other properties to specify multiple images or to rearrange the images and text on the button. *TSpeedButton* also introduces properties that allow speed buttons to work together as a group. Speed buttons are commonly grouped in panels to create specialized tool bars and tool palettes.

See also *TBitBtn* object, *TButton* object, *TToolButton* object

Properties

The following table lists the properties of *TSpeedButton*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
AllowAllUp	published	False	TSpeedButton
BoundsRect	public		TControl
Caption	published		TControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TControl
ClientRect	public, read-only		TControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
Down	published	False	TSpeedButton

S

Property	Visibility	Default	Documented in
Enabled	published	True	TControl
Flat	published	False	TSpeedButton
Font	published		TControl
Glyph	published		TSpeedButton
GroupIndex	published	0	TSpeedButton
Height	published		TControl
Hint	published		TControl
Layout	published	blGlyphLeft	TSpeedButton
Left	published		TControl
Margin	published	–1	TSpeedButton
Name	published		TControl
NumGlyphs	published	1	TSpeedButton
Owner	public, read-only		TComponent
Parent	public		TControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ShowHint	published		TControl
Spacing	published	4	TSpeedButton
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

AllowAllUp

AllowAllUp specifies whether all speed buttons in the group that contains this speed button can be unselected at the same time.

```
property AllowAllUp: Boolean;
```

Description Use *AllowAllUp* with a group of buttons that belong to the same group. Speed buttons belong to the same group if they have the same value on their *GroupIndex* property. Buttons in the same group behave in a mutually exclusive fashion: when one button in the group is selected (down), all other buttons in the group become unselected (up). If *AllowAllUp* is *True*, all of the speed buttons in a group can be unselected at the same time. Clicking the single selected button in the group will deselect that button without selecting another. If *AllowAllUp* is *False*, exactly one of the buttons in the group must be selected at any time. Clicking a down button won't return the button to its up state.

Changing the value of the *AllowAllUp* property for one speed button in a group changes the *AllowAllUp* value for all buttons in the group.

Note If *GroupIndex* is zero, *AllowAllUp* has no effect.

See also *Down* property, *GroupIndex* property

Down

Down specifies whether the button is selected (down) or unselected (up).

```
property Down: Boolean;
```

Description Read *Down* to determine whether a speed button is selected. The *Down* property only applies if the *GroupIndex* property of the button is nonzero.

When *GroupIndex* is greater than 0, set *Down* to *True* to select a button. When the user clicks on a button in the unselected (up) state, the button is selected and *Down* is set to *True*. When the user clicks on a button in the selected (down) state, if *AllowAllUp* is *True*, the button becomes unselected and *Down* is set to *False*.

At design time, specify which button in a group is the initially selected button by setting the *Down* property of the selected button to *True*.

Note When *GroupIndex* is 0, buttons do not remain in the selected state when clicked.

See also *AllowAllUp* property, *GroupIndex* property

Flat

Flat determines whether the button has a a 3D border that provides a raised or lowered look.

```
property Flat: Boolean;
```

Description Set *Flat* to *True* to remove the raised border when the button is unselected and the lowered border when the button is clicked or selected. When *Flat* is *True*, use separate bitmaps for the different button states to provide visual feedback to the user about the button state.

See also *Glyph* property, *NumGlyphs* property

Glyph

Glyph specifies the bitmap that appears on the speed button.

```
property Glyph: TBitmap;
```

Description Set *Glyph* to a bitmap object that contains the image that should appear on the face of the button. Bring up the Open dialog box from the Object Inspector to choose a bitmap file (with a .BMP extension), or specify a bitmap file at runtime.

Glyph can provide up to four images for the speed button with a single bitmap. *TSpeedButton* displays one of these images depending on the state of the button.

Image position in bitmap	Button state	Description
First	Up	This image appears when the button is unselected. If no other images exist in the bitmap, this image is used for all states.
Second	Disabled	This image usually appears dimmed to indicate that the button can't be selected.
Third	Clicked	This image appears when a button is clicked. If *GroupIndex* is 0, The up state image reappears when the user releases the mouse button.
Fourth	Down	This image appears when a button stays down indicating that it remains selected.

If only one image is present, *TSpeedButton* attempts to represent the other states by altering the image slightly for each state, although the *Down* state is always the same as the *Up* state.

If the bitmap contains multiple images, specify the number of images in the bitmap with the *NumGlyphs* property. All images must be the same size and next to each other in a horizontal row.

Note The lower left hand pixel of the bitmap is reserved for the "transparent" color. Any pixel in the bitmap which matches that lower left hand pixel will be transparent.

See also *Down* property, *GroupIndex* property, *Layout* property, *NumGlyphs* property

GroupIndex

GroupIndex allows speed buttons to work together as a group.

```
property GroupIndex: Integer;
```

Description Set *GroupIndex* to determine how the button behaves when clicked.

When *GroupIndex* is 0, the button behaves independently of all other buttons on the form. When the user clicks such a speed button, the button appears pressed (in its clicked state) and then returns to its normal up state when the user releases the mouse button.

When *GroupIndex* is greater than 0, the button remains selected (in its down state) when clicked by the user. When the user clicks a selected button, it returns to the up state, unless *AllowAllUp* is *False*. Setting the *GroupIndex* property of a single speed button to a value greater than 0 causes the button to behave as a two-state button when *AllowAllUp* is *True*.

Speed buttons with the same *GroupIndex* property value (other than 0), work together as a group. When the user clicks one of these buttons, it remains selected until the user clicks another speed button belonging to the same

group. Speed buttons used in this way can present mutually exclusive choices to the user.

See also *AllowAllUp* property, *Down* property

Layout

Layout determines where the image or text appears on the speed button.

```
type TButtonLayout = (blGlyphLeft, blGlyphRight, blGlyphTop, blGlyphBottom);
property Layout: TButtonLayout;
```

Description Set *Layout* to determine the position of the bitmap specified by the *Glyph* property or the text specified by the *Caption* property. These are the possible values:

Value	Meaning
blGlyphLeft	The image or caption appears near the left side of the button.
blGlyphRight	The image or caption appears near the right side of the button.
blGlyphTop	The image or caption appears near the top of the button.
blGlyphBottom	The image or caption appears near the bottom of the button.

See also *Caption* property, *Glyph* property, *Margin* property, *Spacing* property

Margin

Margin is the number of pixels between the edge of the button and the image or caption drawn on its surface.

```
property Margin: Integer;
```

Description Use *Margin* to specify the indentation of the image specified by the *Glyph* property or the text specified by the *Caption* property. The edges that *Margin* separates depends on the *Layout* property. If *Layout* is *blGlyphLeft*, the margin appears between the left edge of the image or caption and the left edge of the button. If *Layout* is *blGlyphRight*, the margin separates the right edges. If *Layout* is *blGlyphTop*, the margin separates the top edges, and if *Layout* is *blGlyphBottom*, the margin separates the bottom edges.

If *Margin* is –1 then the image or text are centered on the button.

See also *Caption* property, *Glyph* property, *Spacing* property

NumGlyphs

NumGlyphs is the number of images specified in the *Glyph* property.

```
type TNumGlyphs: 1..4;
property NumGlyphs: TNumGlyphs;
```

Description Set *NumGlyphs* to the number of images provided by the bitmap assigned to the *Glyph* property. All images must be the same size and next to each other in a row. The *Glyph* property can provide up to four images.

See also *Glyph* property

Spacing

Spacing determines where the image and text appear on a speed button.

property Spacing: Integer;

Description Set *Spacing* to the number of pixels that should appear between the image specified in the *Glyph* property and the text specified in the *Caption* property.

If *Spacing* is a positive number, its value is the number of pixels between the image and text. If *Spacing* is 0, the image and text appear flush with each other. If *Spacing* is –1, the text appears centered between the image and the button edge.

See also *Caption* property, *Glyph* property, *Margin* property, *Layout* property

Events

The following table lists the events of *TSpeedButton*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDblClick	published	TControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl

Methods

The following table lists the methods of *TSpeedButton*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject

Method	Visibility	Documented in
Click	public	TSpeedButton
ClientToScreen	public	TControl
Create	public	TSpeedButton
DefaultHandler	public	TObject
Destroy	public	TSpeedButton
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTextBuf	public	TControl
GetTextLen	public	TControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
Invalidate	public	TControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Perform	public	TControl
Refresh	public	TControl
RemoveComponent	public	TComponent
Repaint	public	TControl
SafeCallException	public	TComponent
ScreenToClient	public	TControl
SendToBack	public	TControl
SetBounds	public	TControl

Method	Visibility	Documented in
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TControl

Click

Click generates an *OnClick* event.

procedure Click; **override**;

Description Call *Click* to generate an *OnClick* event. If the speed button has an *OnClick* event handler, *Click* allows an application to simulate a click on the button.

See also *OnClick* event

Create

Create creates and initializes an instance of *TSpeedButton*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to create a speed button at runtime. Speed buttons placed on forms at design time are created automatically. After calling the inherited constructor, *Create* initializes the following values:

- *ControlStyle* is set to [*csCaptureMouse, csOpaque, csDoubleClicks*].
- *Height* and *Width* are both set to 25.
- *Layout* is set to *blGlyphLeft*.
- *Margin* is set to –1, and *Spacing* to 4.
- *ParentFont* is set to *True*.

See also *ControlStyle* property, *Height* property, *Layout* property, *Margin* property, *ParentFont* property, *Spacing* property, *TCustomControl.Create* method, *Width* property

Destroy

Destroy destroys an instance of *TSpeedButton*.

destructor Destroy; override;

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the speed button is not already freed, and only then calls *Destroy*.

See also *Free* method

TSplitter

Unit: ExtCtrls | Palette: Additional

TObject • TPersistent • TComponent • TControl • TGraphicControl • TSplitter

TSplitter divides the client area of a form into resizable panes.

Description

Add a splitter to a form between two aligned controls to allow users to resize the controls at runtime. The splitter sits between a control aligned to one of the edges of the form and the controls that fill up the rest of the client area. Give the splitter the same alignment as the control that is anchored to the edge of the form. When the user moves the splitter, it resizes the anchored control. This, in turn, changes the client area of the form, and the controls that fill up the rest of the client area resize accordingly.

Use each control on the form as a separate pane. After each pane is placed, place a splitter with the same alignment to allow that pane to be resized. The last pane to be placed on the form should be client-aligned, so that it resizes automatically to fill up the remaining space after all other panes are resized.

See also *TForm* object

Properties

The following table lists the properties of *TSplitter*:

Property	Visibility	Default	Documented in
Align	published	alLeft	TControl
Beveled	published	True	TSplitter
BoundsRect	public		TControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TControl
ClientRect	public, read-only		TControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent

Property	Visibility	Default	Documented in
Enabled	public	True	TControl
Height	published		TControl
Hint	published		TControl
Left	published		TControl
MinSize	published	30	TSplitter
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	True	TControl
ShowHint	public		TControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	public	True	TControl
Width	published		TControl
WindowProc	public		TControl

Beveled

Beveled determines whether the splitter looks beveled along the edge that moves.

```
property Beveled: Boolean;
```

Description Set *Beveled* to *False* to remove the beveled edges from the splitter. Set *Beveled* to *True* to give the edge a 3D beveled look.

The bevel on a splitter runs in a single direction only: along the edges the user moves to resize neighboring objects. Thus, if the *Align* property is *alLeft* or *alRight*, the vertical edges of the splitter are beveled. If the *Align* property is *alTop* or *alBottom*, the horizontal edges of the splitter are beveled.

See also *Align* property

MinSize

MinSize is the minimum size, in pixels, of the client area the splitter must leave on the form.

```
type NaturalNumber = 1..High(Integer);

property MinSize: NaturalNumber;
```

Description Set *MinSize* to provide a minimum size the splitter must leave when resizing its neighboring control.For example, if the *Align* property is *alLeft*, MinSize is the minimum width of the client area to the right of the splitter. If the *Align* property is *alTop*, *MinSize* is the minimum height of the client area below the splitter.

See also *Align* property

Events

The following table lists the events of *TSplitter*:

Event	Visibility	Documented in
OnMoved	published	TSplitter

OnMoved

OnMoved occurs after the user has retiled the form using the splitter.

```
type TNotifyEvent = procedure (Sender: TObject) of object;
property OnMoved: TNotifyEvent;
```

Description Write an *OnMoved* event handler to take specific action when the user finishes dragging the edge of the neighboring control using the splitter. For example, after resizing a control on the edge of the form, use an *OnMoved* event handler to update the layout of the new client area of the form. Use the *ClientRect* property of the form to determine the new size remaining.

See also *ClientRect* property

Methods

The following table lists the methods of *TSplitter*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
Create	public	TSplitter
DefaultHandler	public	TObject
Destroy	public	TGraphicControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject

Method	Visibility	Documented in
DragDrop	public	TControl
Dragging	public	TControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTextBuf	public	TControl
GetTextLen	public	TControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
Invalidate	public	TControl
MethodAddress	public	TObject
MethodName	public	TObject
MouseDown	protected	TSplitter
MouseMove	protected	TSplitter
MouseUp	protected	TSplitter
NewInstance	public	TObject
Paint	protected	TSplitter
Perform	public	TControl
Refresh	public	TControl
RemoveComponent	public	TComponent
Repaint	public	TControl
SafeCallException	public	TComponent
ScreenToClient	public	TControl
SendToBack	public	TControl
SetBounds	public	TControl
SetTextBuf	public	TControl
Show	public	TControl
StopSizing	protected	TSplitter
Update	public	TControl

Create

Create creates an instance of *TSplitter*.

```
constructor Create(AOwner: TComponent); override;
```

Description Call *Create* to instantiate a splitter at runtime. *TSplitter* objects placed on forms at design time are created automatically.

Create allocates memory for the *TSplitter* object and initializes

- *Beveled* to *True*.
- *MinSize* to *30*.
- *Align* to *alLeft*.
- *Width* to *3*.
- *Cursor* to *crHSplit*.

See also *Align* property, *Beveled* property, *Cursor* property, *MinSize* property, *TGraphicControl.Create* method, *Width* property

MouseDown

MouseDown dynamically locates the control that can be resized by the splitter.

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X, Y: Integer); override;
```

Description After calling the inherited method, *MouseDown* locates the control that it is about to resize. Because this control is located dynamically, an application can delete or add aligned controls beside a splitter at runtime and the splitter will resize whichever control is currently beside it.

See also *TControl.MouseDown* method

MouseMove

MouseMove provides visual feedback of how the neighboring control would be resized.

```
procedure MouseMove(Shift: TShiftState; X, Y: Integer); override;
```

Description After calling the inherited method, *MouseMove* computes the new boundaries of the neighboring control and draws a line where the new edge would be. The control is not actually resized until the mouse is released.

See also *MouseUp* method, *TControl.MouseMove* method

MouseUp

MouseUp resizes the neighboring control when the user releases the mouse.

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X, Y: Integer); override;
```

Description After calling the inherited method, *MouseUp* resizes the neighboring control that was located when the mouse went down. After the control is resized, *MouseUp* generates an *OnMoved* event.

See also *MouseDown* method, *OnMoved* event, *TControl.MouseUp* method

Paint

Paint draws the image of the splitter in response to paint messages.

```
procedure Paint; override;
```

Description When the splitter receives WM_PAINT messages from its *Parent*, the splitter calls the *Paint* method. *Paint* draws the background color of the splitter, and if *Beveled* is *True*, draws a bevel along the moveable edges.

See also *Beveled* property, *Color* property, *Parent* property

StopSizing

StopSizing cleans up after the splitter has resized its neighboring control.

```
procedure StopSizing;
```

Description This protected procedure is called when the user releases the mouse or hits the *Escape* key while using the splitter to resize a control. StopSizing makes sure that the resize line is no longer visible and generates an *OnMoved* event.

See also *OnMoved* event

TStaticText

Unit: StdCtrls | Palette: Additional

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomStaticText • TStaticText

TStaticText is a windowed control that displays text on a form.

Description The *TStaticText* component functions like *TLabel*, except that it descends from *TWinControl* and therefore has a window handle. Use *TStaticText* instead of *TLabel* when the component's accelerator key must belong to a windowed control—for example, on an ActiveX property page.

See also *TCustomStaticText* object, *TLabel* object, *TWinControl* object

Properties

The following table lists the properties of *TStaticText*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
Alignment	published	taLeftJustify	TCustomStaticText
AutoSize	published	True	TCustomStaticText
BorderStyle	published	sbsNone	TCustomStaticText
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Caption	published		TControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
FocusControl	published		TCustomStaticText
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	True	TControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl

Property	Visibility	Default	Documented in
ShowAccelChar	published	True	TCustomStaticText
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Events

The following table lists the events of *TStaticText*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TStaticText:*

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject

Method	Visibility	Documented in
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomStaticText
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TWinControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl

Method	Visibility	Documented in
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

TStatusBar

Unit: ComCtrls | Palette: Win32

TObject • TPersistent • TComponent • TControl • TWinControl • TStatusBar

TStatusBar represents a Windows status bar.

Description The *TStatusBar* component is a row of panels, usually aligned at the bottom of the form, that display information about the application as it runs. Each panel is represented by a *TStatusPanel* object listed in the *Panels* property. The *SimplePanel* property can be used to toggle the status bar at runtime between a single-panel and multiple-panel display.

See also *TStatusPanel* object, *TStatusPanels* object

Properties

The following table lists the properties for *TStatusBar*:

Property	Visibility	Default	Documented in
Align	published	alBottom	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Canvas	public, read-only		TStatusBar
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Panels	published		TStatusBar
Parent	public		TControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ShowHint	published		TControl
Showing	public, read-only		TWinControl
SimplePanel	published		TStatusBar
SimpleText	published		TStatusBar
SizeGrip	published	True	TStatusBar

Property	Visibility	Default	Documented in
TabOrder	public	-1	TWinControl
TabStop	public	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Canvas

Provides runtime access to the status bar's drawing surface.

property Canvas: TCanvas;

Description The read-only *Canvas* property provides access to the status bar's drawing surface that you can use when implementing a handler for the *OnDrawPanel* event.

See also *TCanvas* object, *OnDrawPanel* event

Panels

A list of panels (*TStatusPanel*) in the status bar.

property Panels: TStatusPanels;

Description The *Panels* property holds a *TStatusPanels*—that is, a collection of *TStatusPanel* objects. At design time, you can add, remove, or modify panels with the Panels editor. To open the Panels editor, select the *Panels* property in the Object Inspector, then double-click in the Value column to the right or click the ellipsis (...) button.

See also *TStatusPanel* object, *TStatusPanels* object

SimplePanel

Determines whether the status bar displays a single panel or multiple panels.

property SimplePanel: Boolean;

Description If *SimplePanel* is set to true, the status bar consists of a single panel displaying the text in *SimpleText*. If *SimplePanel* is set to false, the status bar displays a separate panel for each item in its *Panels* property. To alter the appearance of the status bar at runtime, change the value of *SimplePanel* programmatically.

See also *Panels* property, *SimpleText* property

SimpleText

Contains the string that is displayed in the status bar when *SimplePanel* is set to True.

```
property SimpleText: string;
```

Description *SimpleText* contains a text string to be displayed in the status panel when *SimplePanel* is True.

See also *SimplePanel* property

SizeGrip

Determines whether the status bar is resizable at runtime.

```
property SizeGrip: Boolean default True;
```

Description If *SizeGrip* is set to True, the status bar has a triangular grip on the lower right corner. The user can resize the status bar by dragging the grip with the mouse.

Note If *Align* is set to *alBottom*, *alRight*, or *alClient*, the sizing grip will not work. If the *Alignment* property of the last panel in the status bar is set to *taRightJustify*, the sizing grip will truncate the panel's text.

See also *Align* property, *TStatusPanel.Alignment* property

Events

The following table lists the events for *TStatusBar*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnDrawPanel	published	TStatusBar
OnEndDrag	published	TControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnResize	published	TStatusBar
OnStartDrag	published	TControl

OnDrawPanel

OnDrawPanel occurs when a status panel needs to be redrawn.

```
type TDrawPanelEvent = procedure(StatusBar: TStatusBar; Panel: TStatusPanel;
  const Rect: TRect) of object;

property OnDrawPanel: TDrawPanelEvent;
```

Description The *OnDrawPanel* event occurs when a status panel needs to be redisplayed—for example, when the user resizes the status bar. The *Rect* parameter gives the (new) dimensions of the status panel. *OnDrawPanel* occurs only if the status panel's *Style* property is set to *psOwnerDraw*.

TDrawPanelEvent is the type of the *OnDrawPanel* property.

See also *TStatusPanel.Style* property, *TRect* type

OnResize

OnResize occurs when the status panel is resized.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnResize: TNotifyEvent;
```

Description The *OnResize* event occurs whenever the entire status bar is resized at runtime.

See also *TNotifyEvent* type

Methods

The following table lists the methods for *TStatusBar*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TStatusBar
CreateParented	public	TWinControl
DefaultHandler	public	TObject

Method	Visibility	Documented in
Destroy	public	TStatusBar
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent

Method	Visibility	Documented in
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

Create

Creates and initializes a *TStatusBar* component.

constructor Create(AOwner: TComponent); **override**;

Description The *Create* method creates a status bar along with its canvas and status panels.

See also *TCanvas* object, *TStatusPanels* object

Destroy

Destroys *TStatusBar* and frees its memory.

destructor Destroy; **override**;

Description The *Destroy* method destroys the status bar along with its canvas and status panels.

See also *TCanvas* object, *TStatusPanels* object

TStoredProc

Unit: DBTables — Palette: Data Access

TObject • TPersistent • TComponent • TDataSet • TBDEDataSet • TDBDataSet • TStoredProc

TStoredProc encapsulates a stored procedure on a database server.

Description Use a *TStoredProc* object when a client application must use a stored procedure on a database server. A *stored procedure* is a grouped set of statements, stored as part of a database server's metadata (just like tables,

indexes, and domains), that performs a frequently-repeated, database-related task on the server and passes results to the client.

Note Not all database servers support stored procedures. See a specific server's documentation to determine if it supports stored procedures.

Many stored procedures require a series of input arguments, or parameters, that are used during processing. *TStoredProc* provides a *Params* property that enables an application to set these parameters before executing the stored procedure.

TStoredProc reuses the *Params* property to hold the results returned by a stored procedure. *Params* is an array of values. Depending on server implementation, a stored procedure can return either a single set of values, or a result set similar to the result set returned by a query.

See also *TQuery* object, *TTable* object

Properties

The following table lists the properties of *TStoredProc*:

Property	Visibility	Default	Documented in
Active	published	False	TDataSet
AutoCalcFields	published	True	TDataSet
BOF	public, read-only		TDataSet
Bookmark	public		TDataSet
CacheBlobs	public	True	TBDEDataSet
CachedUpdates	published	False	TBDEDataSet
CanModify	public, read-only		TBDEDataSet
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Database	public, read-only		TDBDataSet
DatabaseName	published		TDBDataSet
DataSource	public, read-only		TDataSet
DBHandle	public, read-only		TDBDataSet
DBLocale	public, read-only		TDBDataSet
DBSession	public, read-only		TDBDataSet
DefaultFields	public, read-only		TDataSet
Designer	public, read-only		TDataSet
DesignInfo	public		TComponent
EOF	public, read-only		TDataSet
ExpIndex	public, read-only		TBDEDataSet

Property	Visibility	Default	Documented in
FieldCount	public, read-only		TDataSet
FieldDefs	public		TDataSet
Fields	public		TDataSet
FieldValues	public		TDataSet
Filter	published		TBDEDataSet
Filtered	published	False	TBDEDataSet
FilterOptions	published	[]	TBDEDataSet
Found	public, read-only		TDataSet
Handle	public, read-only		TBDEDataSet
KeySize	public, read-only		TBDEDataSet
Locale	public, read-only		TBDEDataSet
Modified	public, read-only		TDataSet
Name	published		TDataSet
Overload	published	0	TStoredProc
Owner	public, read-only		TComponent
ParamBindMode	published	pbByName	TStoredProc
ParamCount	public, read-only		TStoredProc
Params	published		TStoredProc
Prepared	public		TStoredProc
Provider	public, read-only		TDBDataSet
RecNo	public		TBDEDataSet
RecordCount	public, read-only		TBDEDataSet
RecordSize	public, read-only		TBDEDataSet
SessionName	published		TDBDataSet
State	public, read-only		TDataSet
StmtHandle	public, read-only		TStoredProc
StoredProcName	published		TStoredProc
Tag	published	0	TComponent
UpdateObject	published		TBDEDataSet
UpdateRecordTypes	public		TBDEDataSet
UpdatesPending	public, read-only		TBDEDataSet
VCLComObject	public		TComponent

Overload

Specifies which Oracle overloaded stored procedure to execute.

property Overload: Word;

Description Use *Overload* to specify which overloaded stored procedure to execute on an Oracle server. An Oracle overloaded stored procedure is one that shares a name with one or more other stored procedures. Oracle distinguishes among overloaded stored procedures by assigning each procedure a unique

numeric identifier. An application can specify this identifier using the *Overload* property.

By default, *Overload* is zero, which assumes there is no procedure overloading. For all servers except Oracle, do not change this value. If an application attempts to access an Oracle overloaded procedure without setting *Overload*, the Borland Database Engine (BDE) accesses the lowest numbered, or first, overloaded procedure on the server.

If *Overload* is 1, the BDE executes the first overloaded procedure on the server. If *Overload* is 2, it executes the second, and son on.

Note While Oracle overloaded procedures share names, their parameter lists are unique. An application must ensure that it passes the correct parameter list to an overloaded procedure.

See also *Params* property

ParamBindMode

Determines the order in which a component's parameters are assigned to the parameter list for the stored procedure on the server.

```
type TParamBindMode = (pbByName, pbByNumber);

property ParamBindMode: TParamBindMode;
```

Description Examine or set *ParamBindMode* to determine the order in which parameters in the *Params* property are matched to the parameters used by the stored procedure on the server. *ParamBindMode* can be one of the following:

Value	Ordering
pbByName	Parameters specified in the *Params* property are matched to identically named parameters on the server. This is the default.
pbByNumber	Parameters in *Params* are assigned one-by-one to the next available parameter on the server (i.e., the first parameter in *Params* is assigned to the first parameter used by the stored procedure, and so on).

Whenever possible, *ParamBindMode* should be *pbByName*. This guarantees that parameters are matched to the correct parameters used by the stored procedure regardless of physical ordering in *Params*. At design time, the names of known parameters appear in the Parameters editor.

In some cases stored procedures on the server (for example, Informix) do not provide the Borland Database Engine (BDE) with parameter names. If parameter names as they exist on the server side are not known at design or runtime, it may be necessary to set *ParamBindMode* to *pbByNumber*. In this case, however, it is necessary to know the correct data type for each parameter in sequence. Otherwise an exception may be raised on execution.

See also *ExecProc* method, *ParamCount* property, *Params* property

ParamCount

Indicates the number of parameters for the stored procedure component.

property ParamCount: Word;

Description Examine *ParamCount* to determine the number of parameters currently stored in the *Params* property.

See also *ExecProc* method, *Params* property, *Prepare* method

Params

Stores the input and output parameters for a stored procedure.

property Params: TParams;

Description Access *Params* at runtime to set input parameter names, values, and data types dynamically (at design time use the Parameters editor to set parameter information). *Params* is an array of parameter values.

An application can also access *Params* after executing a stored procedure to retrieve the output parameters returned to the procedure by the server.

See also *ExecProc* method, *ParamByName* method, *Prepare* method

Prepared

Determines whether or not a stored procedure is prepared for execution.

property Prepared: Boolean;

Description Examine *Prepared* to determine if a stored procedure is already prepared for execution. If *Prepared* is *True*, the stored procedure is prepared, and if *Prepared* is *False*, the procedure is not prepared. A stored procedure must be prepared before it can be executed.

Note Delphi automatically prepares a stored procedure if it is unprepared when the application calls *ExecProc*. After execution, Delphi automatically unprepares the stored procedure. If a procedure will be executed a number of times, it is more efficient for the application to prepare the stored procedure once, and unprepare it when it is no longer needed.

An application can change the current setting of *Prepared* to prepare or unprepare a stored procedure. If *Prepared* is *True*, setting it to *False* calls the *Unprepare* method to unprepare the stored procedure. If *Prepared* is *False*, setting it to *True* calls the *Prepare* method to prepare the procedure. Generally, however, it is better programming practice to call *Prepare* and *Unprepare* directly. These methods automatically update the *Prepared* property.

See also *ExecProc* method, *Prepare* method, *Unprepare* method

StmtHandle

Identifies the Borland Database Engine (BDE) statement handle for the stored procedure.

```
property StmtHandle: HDBIStmt;
```

Description Retrieve *StmtHandle* if an application makes a direct call to the BDE, bypassing the methods of *TStoredProc*. Some BDE API calls require a statement handle as a parameter. Under all other circumstances an application does not need to access this property.

StoredProcName

Identifies the name of the stored procedure on the server for which this object is an encapsulation.

```
property StoredProcName: string;
```

Description Set *StoredProcName* to specify the name of the stored procedure to call on the server. If *StoredProcName* does not match the name of an existing stored procedure on the server, then when the application attempts to prepare the procedure prior to execution, an exception is raised.

See also *Execute* method, *Prepare* method

Events

The following table lists the events for *TStoredProc*:

Event	Visibility	Documented in
AfterCancel	published	TDataSet
AfterClose	published	TDataSet
AfterDelete	published	TDataSet
AfterEdit	published	TDataSet
AfterInsert	published	TDataSet
AfterOpen	published	TDataSet
AfterPost	published	TDataSet
AfterScroll	published	TDataSet
BeforeCancel	published	TDataSet
BeforeClose	published	TDataSet
BeforeDelete	published	TDataSet
BeforeEdit	published	TDataSet
BeforeInsert	published	TDataSet
BeforeOpen	published	TDataSet
BeforePost	published	TDataSet
BeforeScroll	published	TDataSet

Event	Visibility	Documented in
OnCalcFields	published	TDataSet
OnDeleteError	published	TDataSet
OnEditError	published	TDataSet
OnFilterRecord	published	TDataSet
OnNewRecord	published	TDataSet
OnPostError	published	TDataSet
OnServerYield	published	TBDEDataSet
OnUpdateError	published	TBDEDataSet
OnUpdateRecord	published	TBDEDataSet

Methods

The following table lists the methods of *TStoredProc*:

Method	Visibility	Documented in
ActiveBuffer	public	TDataSet
Append	public	TDataSet
AppendRecord	public	TDataSet
ApplyUpdates	public	TBDEDataSet
Assign	public	TPersistent
BookmarkValid	public	TBDEDataSet
Cancel	public	TBDEDataSet
CancelUpdates	public	TBDEDataSet
CheckBrowseMode	public	TDataSet
CheckOpen	public	TDBDataSet
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClearFields	public	TDataSet
Close	public	TDataSet
CommitUpdates	public	TBDEDataSet
CompareBookmarks	public	TBDEDataSet
ConstraintCallBack	public	TBDEDataSet
ControlsDisabled	public	TDataSet
CopyParams	public	TStoredProc
Create	public	TStoredProc
CursorPosChanged	public	TDataSet
DefaultHandler	public	TObject

Method	Visibility	Documented in
Delete	public	TDataSet
DescriptionsAvailable	public	TStoredProc
Destroy	public	TStoredProc
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableConstraints	public	TBDEDataSet
DisableControls	public	TDataSet
Dispatch	public	TObject
Edit	public	TDataSet
EnableConstraints	public	TBDEDataSet
EnableControls	public	TDataSet
ExecProc	public	TStoredProc
FetchAll	public	TBDEDataSet
FieldAddress	public	TObject
FieldByName	public	TDataSet
FindComponent	public	TComponent
FindField	public	TDataSet
FindFirst	public	TDataSet
FindLast	public	TDataSet
FindNext	public	TDataSet
FindPrior	public	TDataSet
First	public	TDataSet
Free	public	TObject
FreeBookmark	public	TDataSet
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetBookmark	public	TDataSet
GetCurrentRecord	public	TBDEDataSet
GetFieldList	public	TDataSet
GetFieldNames	public	TDataSet
GetIndexInfo	public	TBDEDataSet
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetResults	public	TStoredProc
GotoBookmark	public	TDataSet
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject

Method	Visibility	Documented in
Insert	public	TDataSet
InsertComponent	public	TComponent
InsertRecord	public	TDataSet
InstanceSize	public	TObject
IsEmpty	public	TDataSet
IsLinkedTo	public	TDataSet
Last	public	TDataSet
Locate	public	TBDEDataSet
Lookup	public	TBDEDataSet
MethodAddress	public	TObject
MethodName	public	TObject
MoveBy	public	TDataSet
NewInstance	public	TObject
Next	public	TDataSet
Open	public	TDataSet
ParamByName	public	TStoredProc
Post	public	TDataSet
Prepare	public	TStoredProc
Prior	public	TDataSet
Refresh	public	TDataSet
RemoveComponent	public	TComponent
Resync	public	TDataSet
RevertRecord	public	TBDEDataSet
SafeCallException	public	TComponent
SetFields	public	TDataSet
Translate	public	TDataSet
UnPrepare	public	TStoredProc
UpdateCursorPos	public	TDataSet
UpdateRecord	public	TDataSet
UpdateStatus	public	TBDEDataSet

CopyParams

Copies a stored procedure's parameters into another parameter list.

procedure CopyParams(Value: TParams);

Description Call *CopyParams* to copy this stored procedure's parameters into a separate parameter list object. *Value* is the parameter list into which to assign this stored procedure's parameters. *Value* can be the parameter list of another stored procedure. For example:

```
TStoredProc1.CopyParams(TStoredProc2.Params);
```

If the stored procedure is not prepared when an application calls *CopyParams*, *CopyParams* calls *Prepare* before assigning the parameters to the target parameters list, and then calls *UnPrepare* to return the stored procedure to its previous state.

See also *Params* property, *Prepare* method, *UnPrepare* method

Create

Creates an instance of a stored procedure component.

```
constructor Create(AOwner: TComponent);
```

Description Call *Create* to instantiate a stored procedure declared in an application. *Create* calls its inherited *Create* constructor, creates an empty parameter list for the newly instantiated stored procedure, and initializes its parameter, server, and record buffers to **nil**.

See also *Destroy* method

DescriptionsAvailable

Determines if parameter information is available from the stored procedure on the server.

```
function DescriptionsAvailable: Boolean;
```

Description Call *DescriptionsAvailable* to determine if a stored procedure on a server can return parameter information to the application. If *DescriptionsAvailable* returns *True*, parameter information is available from the server. If *DescriptionsAvailable* returns *False*, parameter information is not available.

Note If *DescriptionsAvailable* returns *False*, the application must still provide correct parameter information to the stored procedure on the server. It is up to the application developer to know the necessary parameters at design time.

See also *ExecProc* method, *GetResults* method, *Params* property

S

Destroy

Destroys the instance of a stored procedure.

```
destructor Destroy;
```

Description Do not call *Destroy* directly. Instead call *Free* to verify that the stored procedure is not already freed before calling *Destroy*. *Destroy* disconnects from the server, frees the parameter list, and calls its inherited *Destroy* destructor.

See also *Free* method, *Params* property

ExecProc

Executes the stored procedure on the server.

```
procedure ExecProc;
```

Description Call *ExecProc* to execute a stored procedure on the server. Before calling *ExecProc*:

1. Provide any input parameters in the *Params* property. At design time, a developer can provide parameters using the Parameters editor. At runtime an application must access *Params* directly.
2. Call *Prepare* to bind the parameters.

If a stored procedure returns output parameters, they are stored in the *Params* property when *ExecProc* returns control to the application. An application can access the output parameters by indexing into the *Params* list, or by using the *ParamByName* method.

Tip If an application is only interested in the result set returned by a stored procedure, call *StoredProc.Open* or set its *Active* property to *True*.

See also *ParamByName* method, *Params* property, *Prepare* method

GetResults

Returns the output parameter values from a Sybase stored procedure.

```
procedure GetResults;
```

Description Call *GetResults* to force a Sybase stored procedure to return its result set, if any, to the client. For stored procedures on all other servers, result sets are automatically flushed to the client, but Sybase stored procedures do not return a result set until the cursor is positioned at the end of the set. *GetResults* forces the cursor to the end of the set.

If a Sybase stored procedure returns a result set and output parameters in response to *GetResults*, the output parameters are stored in the *Params* property. An application can access the output parameters by indexing into the *Params* list, or by using the *ParamByName* method.

Tip If an application is only interested in the result set returned by a stored procedure, call *StoredProc.Open* or set its *Active* property to *True*.

See also *ExecProc* method, *ParamByName* method, *Params* property

ParamByName

Accesses parameter information based on a specified parameter name.

```
function ParamByName(const Value: string): TParam;
```

Description Call *ParamByName* to return parameter information for a specific parameter based on its name. *Value* is the name of the parameter for which to retrieve

information. Typically *ParamByName* is used to set an input parameter's value at runtime, or to retrieve the value of an output parameter. For example, the following statement retrieves the current value of a parameter called "Contact" into an edit box:

```
Edit1.Text := StoredProc1.ParamsByName('Contact').AsString;
```

See also *Params* property

Prepare

Prepares a stored procedure for execution.

procedure Prepare;

Description Call *Prepare* to bind a stored procedure's parameters before calling *ExecProc* to execute the procedure. Prepare readies a stored procedure's parameters, initializes the Borland Database Engine (BDE), and informs the server of the stored procedure's readiness. These steps allocate system resources and optimize the query for server performance.

Note If an application attempts to execute a stored procedure that has not been prepared, Delphi automatically prepares the procedure before executing it, and then unprepares it when execution is complete. If a stored procedure will be executed more than once, it is more efficient for an application to call *Prepare* explicitly once to avoid repeated and unnecessary preparing and unpreparing of the stored procedure, and then call *UnPrepare* when the stored procedure is no longer needed.

See also *ExecProc* method, *Prepared* property, *UnPrepare* method

UnPrepare

Frees the resources allocated for a previously prepared stored procedure.

procedure UnPrepare;

Description Call *UnPrepare* to free the resources allocated for a previously prepared stored procedure on the server and client sides.

See also *ExecProc* method, *Prepare* method

S

TStringField

Unit: DB

TObject • TPersistent • TComponent • TField • TStringField

A *TStringField* object represents a string field in a dataset.

Description A value of a string field is physically stored as a sequence of up to 8192 characters. Common uses for string fields are to store text, such as names and addresses.

TStringField introduces properties to translate between string values and other data types, and to manage language driver conversions. As a descendant of *TField*, *TStringField* inherits many properties, methods, and events that are useful for managing the value and properties of a field in a database.

See also *TDataSet object*, *TMemoField* object

Properties

The following table lists the properties of *TStringField*:

Property	Visibility	Default	Documented in
Alignment	published	taLeftJustify	TField
AsBoolean	public		TStringField
AsCurrency	public		TField
AsDateTime	public		TStringField
AsFloat	public		TStringField
AsInteger	public		TStringField
AsString	public		TStringField
AsVariant	public		TStringField
AttributeSet	public		TField
Calculated	public	False	TField
CanModify	public, read-only		TField
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ConstraintErrorMessage	published		TField
CurValue	public, read-only		TField
CustomConstraint	published		TField
DataSet	public		TField
DataSize	public, read-only		TStringField
DataType	public, read-only		TField
DefaultExpression	published		TField
DesignInfo	public		TComponent
DisplayLabel	published		TField
DisplayName	public, read-only		TField
DisplayText	public, read-only		TField

Property	Visibility	Default	Documented in
DisplayWidth	published		TField
EditMask	published		TField
EditMaskPtr	public, read-only		TField
FieldKind	published	fkData	TField
FieldName	published		TField
FieldNo	public, read-only		TField
ImportedConstraint	published		TField
Index	published		TField
IsIndexField	public, read-only		TField
IsNull	public, read-only		TField
KeyFields	published		TField
Lookup	public		TField
LookupCache	published	False	TField
LookupDataSet	published		TField
LookupKeyFields	published		TField
LookupList	public, read-only		TField
LookupResultField	published		TField
Name	published		TComponent
NewValue	public		TField
Offset	public, read-only		TField
OldValue	public, read-only		TField
Origin	published		TField
Owner	public, read-only		TComponent
ReadOnly	published	False	TField
Required	published	False	TField
Size	published	20	TField
Tag	published	0	TComponent
Text	public		TField
Transliterate	published	True	TStringField
ValidChars	public		TField
Value	public		TStringField
VCLComObject	public		TComponent
Visible	published	True	TField

S

AsBoolean

AsBoolean converts the value of the field to or from a Boolean value.

property AsBoolean: Boolean;

Description Use *AsBoolean* to

- Assign the string field value to a Boolean variable.

- Assign a Boolean value to the string field.

AsBoolean returns *True* on reading the value of the field if its text begins with the letters Y, y, T or t (for Yes or True), and *False* otherwise. Using *AsBoolean* to write a *TStringField*'s value sets the string to T or F.

See also *Value* property

AsDateTime

AsDateTime converts the value of the field to or from a date and time value.

property AsDateTime: TDateTime;

Description Use *AsDateTime* to

- Assign the string field value to a *TDateTime* value.
- Assign a *TDateTime* value to the string field.

Reading *AsDateTime* converts the value of the field to a date-time using the *StrToDateTime* function. Setting *AsDateTime* converts the date-time value to a string using the *DateTimeToStr* function.

See also *DateTimeToStr* function, *StrToDateTime* function, *Value* property

AsFloat

AsFloat converts the value of the field to or from a floating-point value.

property AsFloat: Double;

Description Use *AsFloat* to

- Assign the string field value to a floating-point value.
- Assign a floating-point value to the string field.

For example, this code assigns a floating-point value to a string field value:

```
MyStringField.AsFloat := 7.345;
```

To assign a string field value to a floating-point value, use code similar to this:

```
FloatVariable := MyStringField.AsFloat;
```

Reading *AsFloat* converts the value of the field to a floating-point value using the *StrToFloat* function. Setting *AsFloat* converts the floating-point value to a string using the *FloatToStr* function.

See also *FloatToStr* function, *StrToFloat* function, *Value* property

AsInteger

AsInteger converts the value of the field to or from an integer value.

property AsInteger: Longint;

Description Use *AsInteger* to

- Assign the string field value to an integer value.
- Assign an integer value to the string field.

For example, this code assigns integer value to a string field value:

```
MyStringField.AsInteger := 12;
```

To assign a string field value to an integer value, use code similar to this:

```
IntegerVariable := MyStringField.AsInteger;
```

Reading *AsInteger* converts the value of the field to an integer using the *StrToInt* function. Setting *AsInteger* converts the integer to a string using the *IntToStr* function.

See also *IntToStr function, StrToInt function, Value property*

AsString

AsString gets or sets the value of the field.

property AsString: **string**;

Description String values need no conversion, because the native format of a string field is a string. For string fields, *AsString* is the same as the *Value* property. *AsString* provides a uniform interface that allows applications to get or set string values with generic field components when the internal format of the data is not known.

When setting the *AsString* property, *TStringField* converts the assigned string from ANSI, using the language driver for the dataset, if the *Transliterate* property is *True*. Strings that have more characters than permitted by the value of the *Size* property are truncated when setting *AsString*.

See also *Size* property, *Transliterate* property, *Value* property

AsVariant

AsVariant converts the value of a string field to or from a variant value.

property AsVariant: Variant;

Description Use *AsVariant* to

- Assign the string field value to variant variable.
- Assign a variant value to the string field.

Using variants is useful when both the field type and the type of the value to be assigned to the field are unknown.

See also *AsBoolean* property, *AsDateTime* property, *AsFloat* property, *AsInteger* property, *AsString* property, *EDatabaseError* object

DataSize

DataSize is the amount of memory that is needed to store the value of a string field in its native database format.

property DataSize: Word;

Description Check *DataSize* to determine the number of bytes required to store a string field component's value. Use *DataSize* to determine the size needed for a buffer when working with the *GetData* and *SetData* methods.

For string fields, *DataSize* is one more than the value of the *Size* property. The characters in the string can take up to *Size* bytes, and the extra byte is the null character at the end.

See also *GetData* method, *SetData* method, *Size* property

Transliterate

Transliterate determines whether to convert the value of the string field between the ANSI character set and the character set identified by the dataset, when reading from or writing to the physical database tables.

property Transliterate: Boolean;

Description Use *Transliterate* when the physical database table identified by the dataset does not use an ANSI language driver and the data may contain extended ASCII characters.

When *Transliterate* is *True*, the *AnsiToNative* function is called to translate the ANSI characters to the character set used by the dataset when the string field value is set. When the string field value is read, the *NativeToAnsi* function is called to translate the characters in the database table to the ANSI character set.

See also *AnsiToNative* function, *DataSet* property, *NativeToAnsi* procedure, *TDataSet.Locale* property

Value

Value is the actual data in the string field.

property Value: **string**;

Description Use *Value* to read data directly from and write data directly to a string field component at runtime. Using the *Value* property is the same as using the *AsString* property. Use *Value* when you know the field component is a string field. Use *AsString* when working with a generic field component.

When setting the *Value* property, *TStringField* converts the assigned string from ANSI, using the language driver for the dataset, if the *Transliterate* property is *True*. Strings that have more characters than permitted by the value of the *Size* property are truncated when setting *Value*.

See also *AsString* property, *Size* property, *Transliterate* property

Events

The following table lists the events of *TStringField*:

Event	Visibility	Documented in
OnChange	published	TField
OnGetText	published	TField
OnSetText	published	TField
OnValidate	published	TField

Methods

The following table lists the methods of *TStringField*:

Method	Visibility	Documented in
Assign	public	TField
AssignValue	public	TField
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TField
Create	public	TStringField
DefaultHandler	public	TObject
Destroy	public	TField
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
FocusControl	public	TField
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetData	public	TField
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject

Method	Visibility	Documented in
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
IsBlob	public	TField
IsValidChar	public	TField
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RefreshLookupList	public	TField
RemoveComponent	public	TComponent
SafeCallException	public	TComponent
SetData	public	TField
SetFieldType	public	TField
Validate	public	TField

Create

Create creates an instance of a *TStringField* object.

constructor Create(AOwner: TComponent); **override**;

Description It is seldom necessary to call *Create* directly, because a string field component is instantiated automatically for all string fields in a dataset.

After calling the inherited constructor, *Create* sets

- *DataType* to *ftString*.
- *Size* to 20.
- *Transliterate* to *True*.

See also *Size* property, *TField.Create* method, *Transliterate* property

TStringGrid

Unit: Grids | Palette: Additional

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomControl • TCustomGrid • TDrawGrid • TStringGrid

TStringGrid is a grid control designed to simplify the handling of strings and associated objects.

Description Add a *TStringGrid* object to a form to present textual data in a tabular format. *TStringGrid* provides many properties to control the appearance of the grid, as well as events and methods that take advantage of the tabular organization of the grid in responding to user actions.

TStringGrid introduces the ability to associate an object with each string in the grid. These objects can encapsulate any information or behavior represented by the strings that are presented to the user.

If the strings to be presented in a grid represent field values from the records in a dataset, use *TDBGrid* instead.

See also *TDBGrid* object

Properties

The following table lists the properties of *TStringGrid*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
BorderStyle	published	bsSingle	TCustomGrid
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Canvas	public, read-only		TCustomControl
Cells	public		TStringGrid
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Col	public		TCustomGrid
ColCount	published	5	TCustomGrid
Color	published	clWindow	TControl
Cols	public		TStringGrid
ColWidths	public		TCustomGrid
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl

Property	Visibility	Default	Documented in
DefaultColWidth	published	64	TCustomGrid
DefaultDrawing	published	True	TCustomGrid
DefaultRowHeight	published	24	TCustomGrid
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
EditorMode	public		TCustomGrid
Enabled	published	True	TControl
FixedColor	published	clBtnFace	TCustomGrid
FixedCols	published	1	TCustomGrid
FixedRows	published	1	TCustomGrid
Font	published		TControl
GridHeight	public, read-only		TCustomGrid
GridLineWidth	published	1	TCustomGrid
GridWidth	public, read-only		TCustomGrid
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
LeftCol	public		TCustomGrid
Name	published		TControl
Objects	public		TStringGrid
Options	published	[goFixedVertLine, goFixedHorzLine, goVertLine, goHorzLine, goRangeSelect]	TCustomGrid
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
Row	public		TCustomGrid
RowCount	published	5	TCustomGrid
RowHeights	public		TCustomGrid
Rows	public		TStringGrid
ScrollBars	published	ssBoth	TCustomGrid
Selection	public		TCustomGrid
ShowHint	published		TControl

Property	Visibility	Default	Documented in
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
TabStops	public		TCustomGrid
Tag	published	0	TComponent
Top	published		TControl
TopRow	public		TCustomGrid
VCLComObject	public		TComponent
Visible	published	True	TControl
VisibleColCount	published, read-only		TCustomGrid
VisibleRowCount	published, read-only		TCustomGrid
Width	published		TControl
WindowProc	public		TControl

Cells

Cells is an array of strings, one string for each cell in the grid.

```
property Cells[ACol, ARow: Integer]: string;
```

Description Use *Cells* to access the string within a particular cell. *ACol* is the column coordinate of the cell, and *ARow* is the row coordinate of the cell. The first row is row zero, and the first column is column zero.

The *ColCount* and *RowCount* property values define the size of the array of strings.

To access the objects associated with the strings in the *Cells* array, use the *Objects* property.

See also *ColCount* property, *Objects* property, *RowCount* property

Cols

Cols is an array of the lists strings with their associated objects for each column.

```
property Cols[Index: Integer]: TStrings;
```

Description Use *Cols* to access all the strings for a single column, along with their associated objects. The number of strings and associated objects is always equal to the value of the *ColCount* property, the number of columns in the grid. The *Index* parameter is the number of the column, where the value of the first column in the grid is zero.

To get all the strings and objects for a row, rather than for a column, use the *Rows* property.

See also *ColCount* property, *Objects* property, *Rows* property

Objects

Objects is an array of objects, one for each cell in the grid.

```
property Objects [ACol, ARow: Integer]: TObject;
```

Description Use the *Objects* property to access the object associated with a particular cell.The *ColCount* and *RowCount* values define the size of the array of objects. *ACol* is the column coordinate of the cell, and *ARow* is the row coordinate of the cell, where the first column is column 0, and the first row is row 0.

Note The string grid does not own the objects in the *Objects* array. Objects added to the *Objects* array still exist even if the string grid is destroyed. They must be explicitly destroyed by the application.

See also *Cells* property, *ColCount* property, *RowCount* property

Rows

Rows is an array of the lists of strings and their associated objects for each row.

```
property Rows[Index: Integer]: TStrings;
```

Description Use *Rows* to access all the strings for a single row, along with their associated objects. The number of strings and associated objects is always equal to the value of the *RowCount* property, the number of rows in the grid. The *Index* parameter is the number of the row, where the value of the first row in the grid is zero.

To get all the strings and objects for a column, rather than for a row, use the *Cols* property.

See also *ColCount* property, *Cols* property, *Objects* property

Events

The following table lists the events of *TStringGrid*:

Event	Visibility	Documented in
OnClick	published	TControl
OnColumnMoved	published	TDrawGrid
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnDrawCell	published	TDrawGrid
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnGetEditMask	published	TDrawGrid

Event	Visibility	Documented in
OnGetEditText	published	TDrawGrid
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnRowMoved	published	TDrawGrid
OnSelectCell	published	TDrawGrid
OnSetEditText	published	TDrawGrid
OnStartDrag	published	TControl
OnTopLeftChanged	published	TDrawGrid

Methods

The following table lists the methods of *TStringGrid*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
CellRect	public	TDrawGrid
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TStringGrid
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TStringGrid
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject

Method	Visibility	Documented in
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
MouseCoord	public	TCustomGrid
MouseToCell	public	TDrawGrid
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent

Method	Visibility	Documented in
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

Create

Create creates an instance of *TStringGrid*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to create an instance of *TStringGrid* at runtime. For string grids placed on forms at design time, *Create* is called automatically.

After calling the inherited constructor, *Create* creates the helper objects used by *TStringGrid* to manage the array of strings and their associated objects.

See also *TCustomGrid.Create* method

Destroy

Destroy destroys an instance of *TStringGrid*.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the string list is not already freed, and only then calls *Destroy*.

Destroy frees the helper objects used to manage the array of strings and their associated objects.

See also *Free* method

TTabControl

Unit: ComCtrls Palette: Win32

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomTabControl • TTabControl

TTabControl is a tab set that has the appearance of notebook dividers.

Description Use *TTabControl* to add a control with multiple tab settings to a form. Unlike a page control, *TTabControl* is not made up of several pages that contain different controls. Instead, *TTabControl* is a single object. When the current tab changes, the tab control must directly update its contents to reflect the change in an *OnChange* event handler.

Disabling a *TTabControl* object disables all tabs because it is one control. To disable individual tabs, use *TPageControl* and disable the individual *TTabSheet* controls that represent the pages.

See also *TPageControl* object

Properties

The following table lists the properties for *TTabControl*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DisplayRect	public, read-only		TCustomTabControl
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl

Property	Visibility	Default	Documented in
HotTrack	published	False	TCustomTabControl
Left	published		TControl
MultiLine	published	False	TCustomTabControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ScrollOpposite	published	False	TCustomTabControl
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabHeight	published	0	TCustomTabControl
TabIndex	published	–1	TCustomTabControl
TabOrder	published	–1	TWinControl
TabPosition	published	tpTop	TCustomTabControl
Tabs	published		TCustomTabControl
TabStop	published	True	TWinControl
TabWidth	published	0	TCustomTabControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Events

The following table lists the events for *TTabControl*:

Event	Visibility	Documented in
OnChange	published	TCustomTabControl
OnChanging	published	TCustomTabControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl

Event	Visibility	Documented in
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods for *TTabControl*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomTabControl
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TCustomTabControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent

Method	Visibility	Documented in
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

TTable

Unit: DBTables | Palette: Data Access

TObject • TPersistent • TComponent • TDataSet • TBDEDataSet • TDBDataSet • TTable

Description

TTable is a dataset component that encapsulates a database table.

Use *TTable* to access data in a single database table using the Borland Database Engine (BDE). *TTable* provides direct access to every record and field in an underlying database table, whether it is from Paradox, dBASE, Access, FoxPro, an ODBC-compliant database, or an SQL database on a remote server, such as InterBase, Oracle, Sybase, MS-SQL Server, Informix, or DB2.

A table component can also work with a subset of records within a database table using ranges and filters.

See also

TQuery object, *TStoredProc* object

Properties

The following table lists the properties of *TTable*:

Property	Visibility	Default	Documented in
Active	published	False	TDataSet
AutoCalcFields	published	True	TDataSet
BOF	public, read-only		TDataSet
Bookmark	public		TDataSet
CacheBlobs	public	True	TBDEDataSet
CachedUpdates	published	False	TBDEDataSet
CanModify	public, read-only		TTable
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
Constraints	published		TDataSet
Database	public, read-only		TDBDataSet
DatabaseName	published		TDBDataSet
DataSource	public, read-only		TTable
DBHandle	public, read-only		TDBDataSet
DBLocale	public, read-only		TDBDataSet
DBSession	public, read-only		TDBDataSet
DefaultFields	public, read-only		TDataSet

Property	Visibility	Default	Documented in
Designer	public, read-only		TDataSet
DesignInfo	public		TComponent
EOF	public, read-only		TDataSet
Exclusive	published	False	TTable
ExpIndex	public, read-only		TBDEDataSet
FieldCount	public, read-only		TDataSet
FieldDefs	public		TDataSet
Fields	public		TDataSet
FieldValues	public		TDataSet
Filter	published		TBDEDataSet
Filtered	published	False	TBDEDataSet
FilterOptions	published	[]	TBDEDataSet
Found	public, read-only		TDataSet
Handle	public, read-only		TBDEDataSet
IndexDefs	public, read-only		TTable
IndexFieldCount	public, read-only		TTable
IndexFieldNames	published		TTable
IndexFields	public		TTable
IndexFiles	published		TTable
IndexName	published		TTable
KeyExclusive	public		TTable
KeyFieldCount	public		TTable
KeySize	public, read-only		TBDEDataSet
Locale	public, read-only		TBDEDataSet
MasterFields	published		TTable
MasterSource	published		TTable
Modified	public, read-only		TDataSet
Name	published		TDataSet
Owner	public, read-only		TComponent
Provider	public, read-only		TDBDataSet
ReadOnly	published	False	TTable
RecNo	public		TBDEDataSet
RecordCount	public, read-only		TBDEDataSet
RecordSize	public, read-only		TBDEDataSet
SessionName	published		TDBDataSet
State	public, read-only		TDataSet
TableLevel	public		TTable
TableName	published		TTable
TableType	published	ttDefault	TTable
Tag	published	0	TComponent
UpdateMode	published	upWhereAll	TDBDataSet
UpdateObject	published		TBDEDataSet

Property	Visibility	Default	Documented in
UpdateRecordTypes	public		TBDEDataSet
UpdatesPending	public, read-only		TBDEDataSet
VCLComObject	public		TComponent

CanModify

Indicates whether an application can insert, edit, or delete data in a table.

property CanModify: Boolean;

Description Check the status of *CanModify* to determine if an application can modify a dataset in any way. If *CanModify* is *True*, the dataset can be modified. If *CanModify* is *False*, the table is read-only.

CanModify is set automatically when an application opens a table. If the *ReadOnly* property of a table component is *True*, then *CanModify* is set to *False*. *CanModify* can also be *False* because

- Another application currently has exclusive write access to the table.
- The table is read-only by database design.

Note Even if *CanModify* is *True*, it is not a guarantee that a user will be able to insert or update records in a table. Other factors may come in to play, for example, SQL access privileges.

See also *Exclusive* property, *ReadOnly* property

Constraints

Indicates whether a database server's constraints are replicated to the dataset.

property Constraints: Boolean;

Description Use *Constraints* to specify whether or not a dataset should import constraints from the database server. By default, *Constraints* is *True*. To prevent a dataset from importing server constraints, set *Constraints* to *False* before opening the dataset.

See also *DisableConstraints* method, *EnableConstraints* method

DataSource

Specifies the data source for the table.

property DataSource: TDataSource;

Description Use the *DataSource* property to determine the data source currently assigned to the table. When an application sets the *DataSet* property for a *DataSource* component to the name of this table component, the *DataSource* property for

the table is automatically updated to the name of the data source component that uses it.

See also *Active* property, *IsLinkedTo* method, *TDataSource* object

Exclusive

Enables an application to gain sole access to a Paradox or dBASE table.

```
property Exclusive: Boolean;
```

Description Use *Exclusive* to prevent other applications from accessing a Paradox or dBASE table while this application is using it. Before opening the table, set *Exclusive* to *True*.

When *Exclusive* is *True*, then when the application successfully opens the table, no other application can access it. If the table for which the application has requested exclusive access is already in use by another application, an exception is raised. To handle such exceptions, write an exception handler.

A table must be closed before changing the setting of the *Exclusive* property. Do not set *Exclusive* to *True* at design time if you also intend to set the *Active* property to *True* at design time. In this case an exception is raised because the table is already in use by the IDE.

Note An application can attempt to set *Exclusive* to *True* for SQL tables, but some SQL servers do not support exclusive table-locking. Others may grant an exclusive read/write lock, but still permit read-only access to other applications. For more information about exclusive locking of database tables for a specific server, see the server's documentation.

See also *Active* property, *Close* method, *Open* method

IndexDefs

Contains information about the indexes for a table.

```
property IndexDefs: TIndexDefs;
```

Description Examine *IndexDefs* for index information. *IndexDefs* is an array of index items, each of which describes an available index for the table. Ordinarily an application accesses or specifies indexes through the *IndexFieldNames* or *IndexFields* properties.

Note The index items in *IndexDefs* may not always reflect the current indexes available for a table. Before examining *IndexDefs*, call its *Update* method to refresh the item list.

See also *IndexFieldNames* property, *TIndexDefs* object, *TIndexDef* object

IndexFieldCount

Indicates the number of fields that comprise the current key.

```
property IndexFieldCount: Integer;
```

Description Examine *IndexFieldCount* to determine the number of fields that comprise the current key. For indexes based on a single column, *IndexFieldCount* returns 1. For multi-column indexes, *IndexFieldCount* indicates the number of fields upon which the index is based.

See also *IndexDefs* property

IndexFieldNames

Lists the columns to use as an index for a table.

```
property IndexFieldNames: string;
```

Description Use *IndexFieldNames* as an alternative method of specifying the index to use for a table. In *IndexFieldNames* specify the name of each column to use as an index for a table. Ordering of column names is significant. Separate names with semicolon.

For Paradox and dBASE any column names specified in *IndexFieldNames* must already be indexed. For SQL-based tables, the specified columns need not be indexed.

Note The *IndexFieldNames* and *IndexName* properties are mutually exclusive. Setting one clears the other.

See also *IndexDefs* property, *IndexName* property

IndexFields

Retrieves or sets a field for an index.

```
property IndexFields: [Index: Integer]: TField;
```

Description *IndexFields* provides a zero-based array of field objects, each of which corresponds to a field in the current index. Index is an ordinal value indicating the position of a field in the index. The first field in the index is *IndexFields[0]*, the second is *IndexFields[1]*, and so on.

Note Do not set *IndexField* directly. Instead use the *IndexFieldNames* property to order datasets on the fly at runtime.

See also *AddIndex* method, *IndexDefs* property, *IndexFieldNames* property, *IndexName* property

IndexFiles

Specifies one or more dBASE index file to use for a dBASE table that uses non-production indexes.

```
property IndexFiles: TStrings;
```

Description Use *IndexFiles* to specify dBASE index files to make available for the table that uses non-production indexes. A dBASE index file can contain multiple indexes. Each index in an index file becomes available as a selection for the *IndexName* property which specifies the actual index to use at any given time.

When index files are added to the list of available indexes, the table component opens them. Insertions and modifications made to the table are maintained in the index files. When files are removed from the list, the table component closes them, and they are no longer maintained.

Note At design time, use the Object Inspector to add or remove index file names for the *IndexFiles* property. At runtime multiple index files can be specified by setting *IndexFiles* to a string that contains a list of semicolon-delimited file names.

See also *IndexFieldNames* property, *IndexName* property

IndexName

Identifies a secondary index for the table.

```
property IndexName: string;
```

Description Use *IndexName* to specify an alternative index for a table. Otherwise, if *IndexName* is empty, a table's sort order is based on its primary index, or, for dBASE tables, on its physical record order.

If *IndexName* contains a valid index name, then that index is used to determine sort order of records. For dBASE tables, an index name supplied to the *IndexName* property must either reside in the table's master index file, or in another index file already specified in the *IndexFiles* property.

Note *IndexFieldNames* and *IndexName* are mutually exclusive. Setting one clears the other.

See also *IndexFieldNames* property, *IndexFiles* property

KeyExclusive

Specifies the upper and lower boundaries for a range.

```
property KeyExclusive: Boolean;
```

Description Use *KeyExclusive* to specify whether or not a range includes or excludes the records that match the specified starting and ending values for a range. By default, *KeyExclusive* is *True*, meaning that matching values are included.

To restrict a range to those records that are greater than the specified starting value and less than the specified ending value, set *KeyExclusive* to *False*.

See also *FindNearest* method, *GotoNearest* method, *SetRange* method, *SetRangeEnd* method, *SetRangeStart* method

KeyFieldCount

Specifies the number of fields to use when conducting a partial key search on a multi-field key.

```
property KeyFieldCount: Integer;
```

Description Use *KeyFieldCount* to limit a search based on a multi-field key to a consecutive sub-set of those fields. *KeyFieldCount* is zero-based. For example, if the primary key for a dataset consists of three-fields, a partial-key search can be conducted using only the first fields in the key by setting *KeyFieldCount* to *0*.

Note Searches are only conducted based on consecutive key fields beginning with the first field in the key. For example if a key consists of three fields, an application can set *KeyFieldCount* to *0* to search on the first field, *1* to search on the first and second fields, or *2* to search on all fields. By default *KeyFieldCount* is initially set to include all fields in a search.

See also *FindKey* method, *FindNearest* method, *GotoKey* method, *GotoNearest* method, *Locate* method

MasterFields

Specifies one or more fields in a master table to link with corresponding fields in this table in order to establish a master-detail relationship between the tables.

```
property MasterFields: string;
```

Description Use *MasterFields* after setting the *MasterSource* property to specify the names of one or more fields to use in establishing a detail-master relationship between this table and the one specified in *MasterSource*.

MasterFields is a string containing one or more field names in the master table. Separate field names with semicolons.

Each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this table for display.

Note At design time, use the Field Link designer to establish the master-detail relationship between two tables.

See also *MasterSource* property

MasterSource

Specifies the name of the data source component for the dataset to use as a master table in establishing a detail-master relationship between this table and another one.

```
property MasterSource: TDataSource;
```

Description Use *MasterSource* to specify the name of the data source component whose *DataSet* property identifies a dataset to use as a master table in establishing a detail-master relationship between this table and another one.

Note At design time choose an available data source from the *MasterSource* property's drop-down list in the Object Inspector.

After setting the *MasterSource* property, specify which fields to use in the master table by setting the *MasterFields* property. At runtime each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this table for display.

Note At design time, use the Field Link designer to establish the master-detail relationship between two tables.

See also *MasterFields* property

ReadOnly

Specifies whether a table is read-only for this application.

```
property ReadOnly: Boolean;
```

Description Use the *ReadOnly* property to prevent users from updating, inserting, or deleting data in the table. By default, *ReadOnly* is *False*, meaning users can potentially alter a table's data.

Note Even if *ReadOnly* is False, users may not be able to modify or add data to a table. Other factors, such as insufficient SQL privileges for the application or its current user may prevent successful alterations.

To guarantee that users cannot modify or add data to a table,

1. Set the *Active* property to *False*.
2. Set *ReadOnly* to *True*.

When *ReadOnly* is *True*, the table's *CanModify* property is *False*.

See also *Active* property, *CanModify* property

TableLevel

Specifies the Borland Database Engine (BDE) driver-dependent table level.

```
property TableLevel: Integer;
```

Description Use *TableLevel* when creating a table to specify that the table associated with this dataset requires a specific level of BDE driver. If *TableLevel* is *0*, the default, this property is ignored.

For more information about BDE driver table level specification, see the online BDE help file, BDE32.HLP.

See also *CreateTable* method, *Database* property, *TableType* property

TableName

Indicates the name of the database table that this component encapsulates.

```
property TableName: TFileName;
```

Description Use *TableName* to specify the name of the database table this component encapsulates. To set *TableName* to a meaningful value, the *DatabaseName* property should already be set. If *DatabaseName* is set at design time, then select a valid table name from the *TableName* drop-down list in the Object Inspector.

Note To set *TableName*, the *Active* property must be *False*.

See also *Active* property, *DatabaseName* property, *TableType* property

TableType

Indicates the database table structure for the table that this component encapsulates.

```
type TTableType = (ttDefault, ttParadox, ttDBase, ttASCII);
property TableType: TTableType
```

Description Use *TableType* to specify the database table structure for a dBASE, Paradox, or ASCII table. *TableType* does not apply to tables for remote SQL servers. *TableType* can be set to any of the following values:

Value	Meaning
ttDefault	(Default) Determine table type based on file extension for the table.
ttParadox	Table is a Paradox table.
ttDBase	Table is a dBASE table.
ttASCII	Table is a text file with comma-delimited, quoted strings for each field

If *TableType* is set to *ttDefault*, the table's file-name extension determines the table type:

Extension	Meaning
DB or none	Paradox table
DBF	dBASE table
TXT	ASCII table

If *TableType* is *ttParadox*, *ttDBase*, or *ttASCII*, the table type is assumed to be of the type specified, regardless of the table's file name extension.

See also *IndexName* property, *TableName* property

Events

The following table lists the events of *TTable*:

Event	Visibility	Documented in
AfterCancel	published	TDataSet
AfterClose	published	TDataSet
AfterDelete	published	TDataSet
AfterEdit	published	TDataSet
AfterInsert	published	TDataSet
AfterOpen	published	TDataSet
AfterPost	published	TDataSet
AfterScroll	published	TDataSet
BeforeCancel	published	TDataSet
BeforeClose	published	TDataSet
BeforeDelete	published	TDataSet
BeforeEdit	published	TDataSet
BeforeInsert	published	TDataSet
BeforeOpen	published	TDataSet
BeforePost	published	TDataSet
BeforeScroll	published	TDataSet
OnCalcFields	published	TDataSet
OnDeleteError	published	TDataSet
OnEditError	published	TDataSet
OnFilterRecord	published	TDataSet
OnNewRecord	published	TDataSet
OnPostError	published	TDataSet
OnServerYield	published	TBDEDataSet
OnUpdateError	published	TBDEDataSet
OnUpdateRecord	published	TBDEDataSet

Methods

The following table lists the methods of *TTable*:

Event	Visibility	Documented in
OnCalcFields	published	TDataSet
OnDeleteError	published	TDataSet
OnEditError	published	TDataSet
OnFilterRecord	published	TDataSet
OnNewRecord	published	TDataSet
OnPostError	published	TDataSet
OnServerYield	published	TBDEDataSet

Event	Visibility	Documented in
OnUpdateError	published	TBDEDataSet
OnUpdateRecord	published	TBDEDataSet

AddIndex

Creates a new index for the table.

```
type
  TIndexOptions = set of (ixPrimary, ixUnique, ixDescending, ixCaseInsensitive, ixExpression);

procedure AddIndex(const Name, Fields: string; Options: TIndexOptions);
```

Description Call *AddIndex* to create a new index for the table associated with a dataset. The index created with this procedure is added to the database table underlying the dataset. *Name* is the name of the new index. *Fields* is a semicolon-delimited list of the fields to include in the index. *Options* is a potentially restricted set of one or more of the following values:

Value	Meaning	Restrictions
ixPrimary	Create a primary index	Cannot be used with dBASE. Cannot create more than one primary index per table.
isUnique	Create a unique index	
ixDescending	Sort in descending alphanumeric order	
ixCaseInsensitive	Ignore case when sorting	Cannot be used with dBASE.
ixExpression	Create an expression index	dBASE only.

Attempting to create an index using options that are not applicable to the table raises an exception.

See also *DeleteIndex* method, *IndexDefs* property, *IndexFiles* property, *IndexName* property

ApplyRange

Apply a range to the dataset.

```
procedure ApplyRange;
```

Description Call *ApplyRange* to cause a range established with *SetRangeStart* and *SetRangeEnd*, or *EditRangeStart* and *EditRangeEnd*, to take effect. When a range is in effect, only those records that fall within the range are available to the application for viewing and editing.

See also *CancelRange* method, *EditRangeEnd* method, *EditRangeStart* method, *SetRange* method, *SetRangeEnd* method, *SetRangeStart* method

BatchMove

Moves records from a dataset into this table.

```
type TBatchMode = (batAppend, batUpdate, batAppendUpdate, batDelete, batCopy);
function BatchMove(ASource: TBDEDataSet; AMode: TBatchMode): Longint;
```

Description Call *BatchMove* to

- Copy records from another table into this table.
- Update records in this table that occur in another table.
- Append records from another table to the end of this table.
- Delete records in this table that occur in another table.

ASource is a dataset component containing the records to import for use in this table. *AMode* is one of the following:

Value	Meaning
batAppend	Append all records from the source table to the end of this table.
batAppendUpdate	Append all records from the source table to the end of this table and update existing records in this table with the same records from the source table.
batCopy	Copy all records from the source table into this table.
batDelete	Delete all records in this table that also appear in the source table.
batUpdate	Update existing records in this table with their counterparts in the source table.

BatchMove returns the number of records operated on.

See also *TBatchMove* object

CancelRange

Removes any ranges currently in effect for the table.

```
procedure CancelRange;
```

Description Call *CancelRange* to remove a range currently applied to a table. Canceling a range reenables access to all records in the dataset.

See also *ApplyRange* method, *EditRangeEnd* method, *EditRangeStart* method, *SetRange* method, *SetRangeEnd* method, *SetRangeStart* method

CloseIndexFile

Closes a specified dBASE non-production index file.

```
procedure CloseIndexFile(const IndexFileName: string);
```

Description Call *CloseIndexFile* to shut down a specified dBASE non-production index file. *IndexFileName* is the name of the file to close, and is either the name of an index file previously opened with a call to *OpenIndexFile* or previously opened because the file's name is listed in the *IndexFiles* property.

Once closed, non-production index files are no longer maintained. Subsequent changes made to the dataset are not reflected in the index.

See also *IndexFiles* property, *OpenIndexFile* method

Create

Creates an instance of a table component.

```
constructor Create(AOwner: TComponent);
```

Description Call *Create* to instantiate a table declared in an application if it was not placed on a form at design time. *Create* calls its inherited *Create* constructor, creates an empty index definitions list, creates an empty data link, and creates an empty list of index files.

See also *TDataLink* object, *TIndexDefs* object, *TIndexFiles* object

CreateTable

Builds a new table using new structure information.

```
procedure CreateTable;
```

Description Call *CreateTable* at runtime to create a table using this dataset's current definition. *CreateTable* overwrites an existing table's structure and data.

If the *FieldDefs* property contains values, these values are used to create field definitions. Otherwise the *Fields* property is used. One or both of these properties must contain values in order to recreate a dataset.

If the *IndexDefs* property contain values, these values are used to create index definitions for the dataset.

See also *AddIndex* method, *DeleteIndex* method, *DeleteTable* method, *EmptyTable* method, *FieldDefs* property, *Fields* property, *IndexDefs* property, *RenameTable* method

DeleteIndex

Deletes a secondary index for the table.

```
procedure DeleteIndex(const Name: string);
```

Description Call *DeleteIndex* to remove a secondary index for a table. *Name* is the name of the index to delete. *DeleteIndex* cannot remove a primary index.

Note To delete an index, an application must first open the table for exclusive access.

See also *AddIndex* method, *Exclusive* property, *IndexDefs* property

DeleteTable

Deletes an existing database table.

```
procedure DeleteTable;
```

Description Call *DeleteTable* to delete an existing database table associated with the table component through its *DatabaseName* and *TableName* properties. A table must be closed before it can be deleted.

Warning Deleting a table erases any data the table contains and destroys the table's structure information.

See also *Active* property, *Close* method, *CreateTable* method, *EmptyTable* method

Destroy

Destroys the instance of a table.

```
destructor Destroy;
```

Description Do not call *Destroy* directly. Instead call *Free* to verify that the table is not already freed before calling *Destroy*. *Destroy* frees the index files list for the table, frees its data link, frees its index definitions, and then calls its inherited *Destroy* destructor.

See also *Create* method, *TDataLink* object, *TIndexDefs* object, *TIndexFiles* object

EditKey

Enables modification of the search key buffer.

```
procedure EditKey;
```

Description Call *EditKey* to put the dataset in *dsSetKey* state while preserving the current contents of the current search key buffer. To determine current search keys, you can use the *IndexFields* property to iterate over the fields used by the current index.

EditKey is especially useful when performing multiple searches where only one or two field values among many change between each search.

See also *Fields* property, *FindKey* method, *GotoKey* method, *GotoNearest* method, *SetKey* method, *State* property

EditRangeEnd

Enables changing the ending value for an existing range.

```
procedure EditRangeEnd;
```

Description Call *EditRangeEnd* to change the ending value for an existing range. To specify an end range value, call *FieldByName* after calling *EditRangeEnd*. After assigning a new ending value, call *ApplyRange* to activate the modified range.

Note With Paradox or dBASE tables, *EditRangeEnd* works only on indexed fields. With SQL databases, *EditRangeEnd* also works with any columns specified in the *IndexFieldNames* property.

See also *ApplyRange* method, *CancelRange* method, *EditRangeStart* method, *FieldByName* method, *IndexFieldNames* property, *KeyExclusive* property, *SetRange* method, *SetRangeEnd* method, *SetRangeStart* method

EditRangeStart

Enables changing the starting value for an existing range.

```
procedure EditRangeStart;
```

Description Call *EditRangeStart* to change the starting value for an existing range. To specify a start range value, call *FieldByName* after calling *EditRangeStart*. After assigning a new ending value, call *ApplyRange* to activate the modified range.

Note With Paradox or dBASE tables, *EditRangeStart* works only on indexed fields. With SQL databases, *EditRangeStart* also works with any columns specified in the *IndexFieldNames* property.

See also *ApplyRange* method, *CancelRange* method, *EditRangeEnd* method, *FieldByName* method, *IndexFieldNames* property, *SetRange* method, *SetRangeEnd* method, *SetRangeStart* method

EmptyTable

Deletes all records from the table.

```
procedure EmptyTable;
```

Description The *EmptyTable* method deletes all records from the database table specified by the *DatabaseName* and *TableName* properties.

Note Deletion of records can fail if the user lacks sufficient privileges to perform the delete operation.

See also *DeleteTable* method

FindKey

Searches for a record containing specified field values.

```
function FindKey(const KeyValues: array of const): Boolean;
```

Description Call *FindKey* to search for a specific record in a dataset. *KeyValues* contains a comma-delimited array of field values, called a *key*. Each value in the key can

be a literal, a variable, a null, or **nil**. If the number of values passed in *KeyValues* is less than the number of columns in the index used for the search, the missing values are assumed to be null.

For Paradox and dBASE tables, the key must always be an index, which can be specified in the *IndexName* property. If *IndexName* is empty, the table's primary index.

For SQL tables, the key may correspond to a specified index in *IndexName*, or to a list of field names in the *IndexFieldNames* property.

If a search is successful, *FindKey* positions the cursor on the matching record and returns *True*. Otherwise the cursor is not moved, and *FindKey* returns *False*.

See also *FindNearest* method, *GotoKey* method, *GotoNearest* method, *IndexName* property, *IndexFieldNames* property, *KeyExclusive* property, *Locate* method

FindNearest

Moves the cursor to the record that most closely matches a specified set of key values.

```
procedure FindNearest(const KeyValues: array of const);
```

Description Call *FindNearest* to move the cursor to a specific record in a dataset or to the first record in the dataset that is greater than the values specified in the *KeyValues* parameter. *KeyValues* contains a comma-delimited array of field values, called a *key*. If the number of values passed in *KeyValues* is less than the number of columns in the index used for the search, the missing values are assumed to be null.

For Paradox and dBASE tables, the key must always be an index, which can be specified in the *IndexName* property. If *IndexName* is empty, the table's primary index.

For SQL tables, the key may correspond to a specified index in *IndexName*, or to a list of field names in the *IndexFieldNames* property.

FindNearest positions the cursor either on a record that exactly matches the search criteria, or on the first record whose values are greater than those specified in the search criteria. *KeyExclusive* also affects the boundary conditions of ranges and will affect the record selected by *FindNearest*.

See also *FindKey* method, *GotoKey* method, *GotoNearest* method, *IndexName* property, *IndexFieldNames* property, *KeyExclusive* property, *Locate* method

GetIndexNames

Retrieves a list of available indexes for a table.

```
procedure GetIndexNames(List: TStrings);
```

Description Call *GetIndexNames* to retrieve a list of all available indexes for a table. *List* is a string list object, created and maintained by the application, into which to retrieve the index names.

See also *IndexName* property

GotoCurrent

Synchronizes the current record for this table with the current record of a specified table component.

```
procedure GotoCurrent(Table: TTable);
```

Description Call *GotoCurrent* to synchronize the cursor position for this table based on the cursor position in another dataset that uses a different data source component, but which is connected to the same underlying database table. *Table* is the name of the table component whose cursor position to use for synchronizing.

Note This procedure works only for table components that have the same *DatabaseName* and *TableName* properties. Otherwise an exception is raised.

GotoCurrent is mainly for use in applications that have two table components that are linked to the same underlying database table through different data source components. It enables an application to ensure that separate views of the data appear to be linked.

See also *DatabaseName* property, *TableName* property, *TDataSource* object

GotoKey

Moves the cursor to a record specified by the current key.

```
function GotoKey: Boolean;
```

Description Use *GotoKey* to move to a record specified by key values assigned with previous calls to *SetKey* or *EditKey* and actual search values indicated in the *Fields* property.

If *GotoKey* finds a matching record, it positions the cursor on the record and returns *True*. Otherwise the current cursor position remains unchanged, and *GotoKey* returns *False*.

See also *Fields* property, *FindKey* method, *FindNearest* method, *GotoNearest* method, *SetKey* method, *State* property

GotoNearest

Moves the cursor to the record that most closely matches the current key.

```
procedure GotoNearest;
```

Description Call *GotoNearest* to position the cursor on the record that is either the exact record specified by the current key values in the key buffer, or on the first record whose values exceed those specified.

Note *KeyExclusive* determines which records are considered part of a search range.

Before calling *GotoNearest*, an application must specify key values by calling *SetKey* or *EditKey* to put the dataset is *dsSetKey* state, and then use *FieldByName* to populate the key buffer property with search values.

See also *FieldByName* method, *FindKey* method, *FindNearest* method, *GotoKey* method, *SetKey* method, *State* property

LockTable

Locks a Paradox or dBASE table.

```
type TLockType = (ltReadLock, ltWriteLock);
procedure LockTable(LockType: TLockType);
```

Description Call *LockTable* to lock a Paradox or dBASE table to prevent other applications from placing a particular type of lock on the table. *LockType* specifies the lock requested by this application.

Requesting a read lock prevents other applications from reading a table. Requesting a write lock prevents other application from writing to a table.

An application can request both types of lock by calling *LockTable* twice, once for each lock type.

See also *UnLockTable* method

OpenIndexFile

Opens a specified dBASE non-production index file.

```
procedure OpenIndexFile(const IndexFileName: string);
```

Description Call *OpenIndexFile* to open a specified dBASE non-production index file. *IndexFileName* is the name of the file to open. *OpenIndexFile* opens the file and adds its name to the list of index files in the IndexFiles property.

While a non-production index file is open, it is maintained.

See also *IndexFiles* property, *CloseIndexFile* method

RenameTable

Renames the Paradox or dBASE table associated with this table component.

```
procedure RenameTable(const NewTableName: string);
```

Description Call *RenameTable* to give a new name to the Paradox or dBASE table underlying this table component. *RenameTable* renames the table and any support files (for example, Paradox .DB and .MB files).

See also *CreateTable* method, *DeleteTable* method, *EmptyTable* method, *LockTable* method, *UnlockTable* method

SetKey

Enables setting of keys and ranges for a dataset prior to a search.

```
procedure SetKey;
```

Description Call *SetKey* to put the dataset into *dsSetKey* state and clear the current contents of the key buffer. The *FieldByName* method can be used to supply th a new set of search values prior to conducting a search.

Note To modify an existing key or range, call *EditKey*.

See also *EditKey* property, *SetRange* method, *SetRangeEnd* method, *SetRangeStart* method, *State* property

SetRange

Sets the starting and ending values of a range, and applies it.

```
procedure SetRange(const StartValues, EndValues: array of const);
```

Description Call *SetRange* to specify a range and apply it to the dataset. *StartValues* indicates the field values that designate the first record in the range. *EndValues* indicates the field values that designate the last record in the range.

SetRange combines the functionality of *SetRangeStart*, *SetRangeEnd*, and *ApplyRange* in a single procedure call. *SetRange* performs the following functions:

1. Puts the dataset into *dsSetKey* state.
2. Erases any previously specified starting range values and ending range values.
3. Sets the start and end range values.
4. Applies the range to the dataset.

If either *StartValues* or *EndValues* has fewer elements than the number of fields in the current index, then the remaining entries are set to NULL.

Note With Paradox or dBASE tables, *EditRangeStart* works only on indexed fields. With SQL databases, *EditRangeStart* also works with any columns specified in the *IndexFieldNames* property.

See also *ApplyRange* method, *CancelRange* method, *EditRangeEnd* method, *EditRangeStart* method, *FieldValues* property, *IndexFieldNames* property, *KeyExclusive* property, *SetRangeEnd* method, *SetRangeStart* method

SetRangeEnd

Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

```
procedure SetRangeEnd;
```

Description Call *SetRangeEnd* to put the dataset into *dsSetKey* state, erase any previous end range values, and set them to NULL. Subsequent key buffer field assignments can be made using the *FieldByName* method to set the ending values for a range.

After assigning end-range values to *FieldValues*, call *ApplyRange* to activate the modified range.

Note With Paradox or dBASE tables, *SetRangeEnd* works only on indexed fields. With SQL databases, *SetRangeEnd* also works with any columns specified in the *IndexFieldNames* property.

See also *ApplyRange* method, *CancelRange* method, *EditRangeEnd* method, *EditRangeStart* method, *FieldByName* property, *IndexFieldNames* property, *KeyExclusive* property, *SetRange* method, *SetRangeStart* method

SetRangeStart

Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.

```
procedure SetRangeStart;
```

Description Call *SetRangeStart* to put the dataset into *dsSetKey* state, erase any previous start range values, and set them to NULL. Subsequent field assignments to the *FieldValues* property indicate the actual set of starting values for a range.

After assigning start-range values to *FieldValues*, call *ApplyRange* to activate the modified range.

Note With Paradox or dBASE tables, *SetRangeEnd* works only on indexed fields. With SQL databases, *SetRangeEnd* also works with any columns specified in the *IndexFieldNames* property.

See also *ApplyRange* method, *CancelRange* method, *EditRangeEnd* method, *EditRangeStart* method, *FieldValues* property, *IndexFieldNames* property, *KeyExclusive* property, *SetRange* method, *SetRangeEnd* method

UnlockTable

Removes a previously applied lock on a Paradox or dBASE table.

```
type TLockType = (ltReadLock, ltWriteLock);

procedure UnlockTable(LockType: TLockType);
```

Description Call *UnlockTable* to remove a lock previously applied to a Paradox or dBASE table to prevent other applications from placing a particular type of lock on the table. *LockType* specifies the lock to remove.

Removing a read lock enables other applications to read a table. Removing a write lock enables other application to write to a table.

An application can remove both types of lock by calling *UnlockTable* twice, once for each lock type.

See also *LockTable* method

TTabSheet

Unit: ComCtrls

TObject • TPersistent • TComponent • TControl • TWinControl • TTabSheet

TTabSheet is an individual page in a *TPageControl* object.

Description Use *TTabSheet* to represent a single page in a tabbed page control. Tab sheets are typically referred to as pages. *TPageControl* maintains an indexed array of its tab sheets in its *Pages* property. Users can click on a tab to activate the tab sheet.

See also *TPageControl* object

Properties

The following table lists the properties for *TTabSheet*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
Caption	published		TControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl

Property	Visibility	Default	Documented in
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
PageControl	public		TTabSheet
PageIndex	published		TTabSheet
Parent	public		TControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabIndex	public, read-only		TTabSheet
TabOrder	public	–1	TWinControl
TabStop	public	False	TWinControl
TabVisible	published	True	TTabSheet
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

PageControl

PageControl is the page control object that contains the tab sheet.

property PageControl: TPageControl;

Description Read *PageControl* to gain access to the page control object that uses the tab sheet. The properties and methods of the *PageControl* object can be used to

locate the selected page, iterate through the other pages in the page control, or change the display properties of the tabs.

Set *PageControl* to remove the tab sheet from its current page control (if any) and insert it into a new page control.

See also *TPageControl* object

PageIndex

PageIndex is the index of the tab sheet in the list of tab sheets maintained by the page control.

```
property PageIndex: Integer;
```

Description Use *PageIndex* to determine where the tab sheet sits in its page control. Each tab sheet in a page control is automatically assigned a *PageIndex* when it is inserted in the page control. The first tab sheet receives a value of 0, the second has a value of 1, and so on. *PageIndex* values are reassigned if tab sheets are deleted or moved.

To access a particular tab sheet given its index value, use the indexed *Pages* property of the *TPageControl* object.

See also *PageControl* property, *TPageControl.Pages* property

TabIndex

TabIndex indicates the position of the tab sheet in the set of visible tabs in a *TPageControl* object.

```
property TabIndex: Integer;
```

Description Read *TabIndex* to determine where the user sees the tab sheet. A value of 0 indicates the first visible tab sheet, a value of 1 indicates the second visible tab sheet, and so on. *TabIndex* is always less than or equal to *PageIndex*. For example, a tab sheet that has a *PageIndex* of 3 will have a *TabIndex* of 2 if one of the previous tab sheets in the page control is not visible.

If a tab sheet's *TabVisible* property is *False*, the *TabIndex* property is –1.

See also *PageIndex* property, *TabVisible* property

TabVisible

TabVisible specifies whether the tab of the *TTabSheet* object appears in its *TPageControl*.

```
property TabVisible Boolean;
```

Description Use *TabVisible* to temporarily remove a tab sheet from a page control. When *TabVisible* is *False*, the tab does not appear in the page control and its *TabIndex* property is –1. Setting *TabVisible* to *True* allows the user to see the tab of the tab sheet again.

See also *TabIndex* property

Events

The following table lists the events for *TTabSheet*:

Event	Visibility	Documented in
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl

Methods

The following table lists the methods for *TTabSheet*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TTabSheet
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TTabSheet
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl

Method	Visibility	Documented in
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl

Method	Visibility	Documented in
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

Create

Create creates and initializes an instance of *TTabSheet*.

```
constructor Create(AOwner: TComponent); override;
```

Description Call *Create* to instantiate a tab sheet at runtime. Tab sheets added to a page controls at design time are created automatically. *Create* calls the inherited *Create* method, then sets the initial values for the tab sheet component.

When creating tab sheets at runtime, use the *PageControl* property to insert them into a page control object.

See also *PageControl* property, *TWinControl.Create* method

Destroy

Destroy destroys an instance of *TTabSheet*.

```
destructor Destroy; override;
```

Description Do not call *Destroy* directly at runtime. Instead, call *Free*. *Free* verifies that the tab sheet is not already freed and only then calls *Destroy*. *Destroy* removes the tab sheet from the page control and then calls the inherited *Destroy* method.

See also *Free* method, *PageControl* property

TTimeField

Unit: DB

TObject • TPersistent • TComponent • TField • TDateTimeField • TTimeField

A *TTimeField* object represents a date field in a dataset.

Description Time fields contain time values.

TTimeField introduces no new properties, events, or methods. It overrides the *DataSize* property and *Create* method to reflect the specifics of a field of type *ftTime*.

As a descendant of *TDateTimeField*, *TTimeField* includes many properties, methods, and events that are useful for managing the value and properties of a temporal field in a database.

See also *TDataSet* object, *TDateField* object

Properties

The following table lists the properties of *TTimeField*:

Property	Visibility	Default	Documented in
Alignment	published	taLeftJustify	TField
AsBoolean	public		TField
AsCurrency	public		TField
AsDateTime	public		TDateTimeField
AsFloat	public		TDateTimeField
AsInteger	public		TField
AsString	public		TDateTimeField
AsVariant	public		TDateTimeField
AttributeSet	public		TField
Calculated	public	False	TField
CanModify	public, read-only		TField
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ConstraintErrorMessage	published		TField
CurValue	public, read-only		TField
CustomConstraint	published		TField
DataSet	public		TField
DataSize	public, read-only		TTimeField
DataType	public, read-only		TField
DefaultExpression	published		TField
DesignInfo	public		TComponent
DisplayFormat	published		TDateTimeField
DisplayLabel	published		TField
DisplayName	public, read-only		TField
DisplayText	public, read-only		TField
DisplayWidth	published		TField
EditMask	published		TField
EditMaskPtr	public, read-only		TField
FieldKind	published	fkData	TField

Property	Visibility	Default	Documented in
FieldName	published		TField
FieldNo	public, read-only		TField
ImportedConstraint	published		TField
Index	published		TField
IsIndexField	public, read-only		TField
IsNull	public, read-only		TField
KeyFields	published		TField
Lookup	public		TField
LookupCache	published	False	TField
LookupDataSet	published		TField
LookupKeyFields	published		TField
LookupList	public, read-only		TField
LookupResultField	published		TField
Name	published		TComponent
NewValue	public		TField
Offset	public, read-only		TField
OldValue	public, read-only		TField
Origin	published		TField
Owner	public, read-only		TComponent
ReadOnly	published	False	TField
Required	published	False	TField
Size	public		TField
Tag	published	0	TComponent
Text	public		TField
ValidChars	public		TField
Value	public		TDateTimeField
VCLComObject	public		TComponent
Visible	published	True	TField

DataSize

DataSize is the amount of memory that is needed to store the value of a time field in its native database format.

property DataSize: Word;

Description Check *DataSize* to determine the number of bytes required to store a time field component's value. Use *DataSize* to determine the size needed for a buffer when working with the *GetData* and *SetData* methods.

For time fields, *DataSize* is always 4. This is the size of time values in the database tables, not the size of the *TDateTime* data type used to store and manipulate time values by *TTimeField*.

See also *GetData* method, *SetData* method, *Value* property

Events

The following table lists the events of *TTimeField*:

Event	Visibility	Documented in
OnChange	published	TField
OnGetText	published	TField
OnSetText	published	TField
OnValidate	published	TField

Methods

The following table lists the methods of *TTimeField*:

Method	Visibility	Documented in
Assign	public	TField
AssignValue	public	TField
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TField
Create	public	TTimeField
DefaultHandler	public	TObject
Destroy	public	TField
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
FocusControl	public	TField
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetData	public	TField
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent

Method	Visibility	Documented in
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
IsBlob	public	TField
IsValidChar	public	TField
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RefreshLookupList	public	TField
RemoveComponent	public	TComponent
SafeCallException	public	TComponent
SetData	public	TField
SetFieldType	public	TField
Validate	public	TField

Create

The *Create* method creates and initializes an instance of *TTimeField*.

```
constructor Create(AOwner: TComponent); override;
```

Description It is seldom necessary to call *Create* directly, because a time field object is instantiated automatically for all time fields in a dataset.

After calling the inherited constructor, *Create* sets the *DataType* property to *ftTime*.

See also *DataType property*, *TField.Create* method

TTimer

Unit: ExtCtrls Palette: System

TObject • TPersistent • TComponent • TTimer

TTimer is an object that encapsulates the Windows API timer functions.

Description *TTimer* is a component that is used to simplify calling the Windows API timer functions *SetTimer* and *KillTimer*, and to simplify processing the WM_TIMER messages. Use one timer component for each timer in the application.

Timer properties and methods affect the functionality of the timer by providing information for the timer event. This information includes the

timer interval, which corresponds to the parameter for the Windows API *SetTimer* function. The actual execution of the timer occurs through its *OnTimer* event.

Caution Limitations on the total number of timers system-wide are system dependent.

Properties

The following table lists the properties of *TTimer*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DesignInfo	public		TComponent
Enabled	published	True	TTimer
Interval	published	1000	TTimer
Name	published		TComponent
Owner	public, read-only		TComponent
Tag	published	0	TComponent
VCLComObject	public		TComponent

Enabled

Enabled controls whether the timer responds to timer events.

```
property Enabled: Boolean;
```

Description Use *Enabled* to enable or disable the timer. If *Enabled* is *True*, the timer responds normally. If *Enabled* is *False*, the timer ignores the *OnTimer* event. The default is *True*.

See also *OnTimer* event

Interval

Interval determines the amount of time, in milliseconds, that passes before the timer component initiates another *OnTimer* event.

```
property Interval: Cardinal;
```

Description *Interval* determines how frequently the *OnTimer* event occurs. Each time the specified interval passes, the *OnTimer* event occurs.

Use *Interval* to specify any cardinal value as the interval between *OnTimer* events. The default value is 1000 (one second).

Note A 0 value is valid, however the timer won't call an *OnTimer* event for a value of 0.

See also *OnTimer* event

Events

The following table lists the events of *TTimer*:

Event	Visibility	Documented in
OnTimer	published	TTimer

OnTimer

OnTimer occurs when a specified amount of time, determined by the *Interval* property, has passed.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnTimer: TNotifyEvent;
```

Description Use *OnTimer* to write an event handler to execute an action at regular intervals.

The *Interval* property of a timer determines how frequently the *OnTimer* event occurs. Each time the specified interval passes, the *OnTimer* event occurs.

Example The code in this *OnTimer* event handler moves a ball, the shape component (*TShape*) slowly across a form.

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  Shape1.Shape := stCircle;
  Shape1.Left := Shape1.Left + 1;
end;
```

See also *Interval* property

Methods

The following table lists the methods of *TTimer*:

Method	Visibility	Documented in
Assign	public	TPersistent
ClassInfo	public	TObject

Method	Visibility	Documented in
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TTimer
DefaultHandler	public	TObject
Destroy	public	TTimer
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Create

Create instantiates a timer object.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to instantiate a timer at runtime. For timers created at design time, *Create* is called automatically.

Create allocates memory for a timer, and calls the inherited *Create* method. Then it

- Initializes the *Enabled* property to *True* .
- Initializes the *Interval* property to 1000.
- Allocates a window handle for the timer passing it the timer window procedure which processes the WM_TIMER messages.

See also *Enabled* property, *Interval* property

Destroy

Destroy disposes of a timer object.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, an application should call *Free*. Free verifies that the list box object has not already been freed before it calls *Destroy*.

Destroy deactivates the timer by setting *Enabled* to *False*. Then it kills the timer by calling the Windows API *KillTimer* function and deallocates the timer window handle before calling the inherited *Destroy*.

See also *Free* method

TToolBar

Unit: ComCtrls | Palette: Win32

TObject • TPersistent • TComponent • TControl • TWinControl • TToolWindow • TToolBar

TToolBar manages tool buttons and other controls, arranging them in rows and automatically adjusting their sizes and positions.

Description *TToolBar* is a container for tool buttons (*TToolButton*). It provides an easy way to arrange and manage visual controls.

- All tool buttons on a tool bar maintain a uniform width and height.
- Other controls can sit on a tool bar. These controls (which are held in place by invisible tool buttons) maintain a uniform height.
- Controls can automatically wrap around and start a new row when they do not fit horizontally on the tool bar.
- The *Flat* property allows the background to show through the tool bar and gives pop-up borders to the tool buttons.
- Spaces and dividers (which are in fact specially configured tool buttons) can group controls on the tool bar both visually and functionally.

Typically, the tool buttons correspond to items in an application's menu and give the user more direct access to the application's commands.

See also *Buttons* property, *TToolButton* object

Properties

The following table lists the properties of *TToolBar*:

Property	Visibility	Default	Documented in
Align	published	alTop	TControl
AutoSize	published		TToolBar
BorderWidth	published	0	TToolWindow
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ButtonCount	public, read-only		TToolBar
ButtonHeight	published	22	TToolBar
Buttons	public, read-only		TToolBar
ButtonWidth	published	23	TToolBar
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DisabledImages	published		TToolBar
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
EdgeBorders	published	[ebLeft, ebTop, ebRight, ebBottom]	TToolWindow
EdgeInner	published	esRaised	TToolWindow
EdgeOuter	published	esLowered	TToolWindow
Flat	published	False	TToolBar
Font	published		TControl
Handle	public, read-only		TWinControl

Property	Visibility	Default	Documented in
Height	published	32	TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
HotImages	published		TToolBar
Images	published		TToolBar
Indent	published	0	TToolBar
Left	published		TControl
List	published	False	TToolBar
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentColor	published	True	TControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
RowCount	public, read-only		TToolBar
ShowCaptions	published	False	TToolBar
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl
Wrapable	published	True	TToolBar

AutoSize

Changes the tool bar's height to accommodate the controls in it.

property AutoSize: Boolean;

Description If *AutoSize* is set to *True*, the height of the tool bar adjusts automatically so that the controls fit in it with no vertical margin.

See also *Wrapable* property

ButtonCount

The number of buttons in the tool bar.

property ButtonCount: Integer;

Description *ButtonCount* returns the number of buttons in the *Buttons* array.

See also *Buttons* property

ButtonHeight

The height of the controls in the tool bar.

property ButtonHeight: Integer;

Description *ButtonHeight* represents the height, in pixels, of the controls in the tool bar.

See also *ButtonWidth* property

Buttons

A list of tool buttons (*TToolButton*) in the tool bar.

property Buttons[Index: Integer]: TToolButton;

Description *Buttons* maintains a list of *TToolButton* instances. All tool buttons that share a *TToolBar* parent have the same height and (except for separators and dividers) the same width. Other controls on a tool bar are held in place by invisible separators, which are automatically created and destroyed.

To add tool buttons to the tool bar at design time, select the tool bar, right-click, and choose New Button. To create a space (separator) between one button and the next, select New Separator. To create a divider between buttons, add a button and set its *Style* property to *tbsDivider*. Other controls may be added to the tool bar directly from the Component palette.

See also *TList* object, *TToolButton* object, *TToolButton.Style* property

ButtonWidth

The width of the tool buttons (*TToolButton*) in the tool bar.

property ButtonWidth: Integer;

Description *ButtonWidth* represents the width, in pixels, of the *TToolButton* components in the tool bar. It has no effect on tool buttons whose *Style* property is set to *tbsSeparator* or *tbsDivider*.

See also *ButtonHeight* property, *Buttons* property, *TToolButton* object, *TToolButton.Style* property

DisabledImages

A list of images that can appear on disabled tool buttons.

property DisabledImages: TImageList;

Description *DisabledImages* contains a list of images that can appear on tool buttons when they are disabled. Each tool button's *ImageIndex* property determines the specific image displayed on it. If *DisabledImages* is unassigned, disabled tool buttons display grayed versions of their regular images (specified by the *Images* property).

See also *HotImages* property, *Images* property, *TImageList* object, *TToolButton.Enabled* property, *TToolButton.ImageIndex* property

Flat

Makes the tool bar transparent and eliminates tool-button borders.

```
property Flat: Boolean;
```

Description If *Flat* is set to *True*, the tool bar and its tool buttons are transparent; background colors and graphics show through them. In addition, when *Flat* is *True*, tool buttons (*TToolButton*) on the tool bar have pop-up borders that appear only when the mouse is passed over the button.

Note If *Flat* is set to *True*, *TToolBar* requires version 4.70 or later of COMCTL32.DLL at both design time and runtime.

See also *Buttons* property, *HotImages* property, *TToolButton* object

HotImages

A list of images that can appear on tool buttons when the mouse points to them.

```
property HotImages: TImageList;
```

Description *HotImages* contains a list of images that can appear on tool buttons when the mouse passes over them at runtime. Each tool button's *ImageIndex* property determines the specific image displayed when the mouse points to it. If *HotImages* is unassigned, tool buttons retain their regular images (specified by the *Images* property) when pointed to.

See also *DisabledImages* property, *Images* property, *TImageList* object, *TToolButton.ImageIndex* property

T

Images

A list of images that can appear on tool buttons.

```
property Images: TImageList;
```

Description *Images* contains a list of images that can appear on the tool bar's tool buttons. Each tool button's *ImageIndex* property determines the specific image displayed on it. If *Images* is unassigned, no images are displayed on the tool buttons.

See also *DisabledImages* property, *HotImages* property, *TImageList* object, *TToolButton.ImageIndex* property

Indent

Creates a margin on the left side of the tool bar.

```
property Indent: Integer;
```

Description *Indent* specifies, in pixels, the margin that separates the left edge of the tool bar from the controls within it.

List

Aligns captions on the right and images on the left in tool buttons.

```
property List: Boolean;
```

Description When *List* is *False*, tool buttons (*TToolButton*) appear in the tool bar with their images (if any) centered and their captions (if visible) underneath. Setting *List* to *True* aligns captions to the right and images to the left.

See also *Images* property, *ShowCaption* property, *TToolButton* object

RowCount

The number of rows in the tool bar.

```
property RowCount: Integer;
```

Description *RowCount* returns the number of rows the tool buttons are arranged in.

See also *Wrapable* property

ShowCaptions

Determines whether text captions are displayed on tool buttons.

```
property ShowCaptions: Boolean;
```

Description If *ShowCaptions* is set to *True*, tool buttons appear in the tool bar with their captions displayed. Setting *ShowCaptions* to *True* may increase the size of the tool buttons.

See also *Image* property, *TToolButton.Caption* property

Wrapable

Enables line-wrap for components in the tool bar.

```
property Wrapable: Boolean;
```

Description If *Wrapable* is *True*, components wrap around and form new rows when they do not fit horizontally in the tool bar.

See also *RowCount* property, *TToolButton.Wrap* property

Events

The following table lists the events of *TToolBar*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDblClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnResize	published	TToolBar
OnStartDrag	published	TControl

OnResize

OnResize occurs when the tool bar is resized.

```
type TNotifyEvent = procedure(Sender: TObject) of object;
property OnResize: TNotifyEvent;
```

Description The *OnResize* event occurs whenever the *TToolBar* to is resized at runtime.

See also *BoundsRect* property

Methods

The following table lists the methods of *TToolBar*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject

Method	Visibility	Documented in
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TToolBar
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TToolBar
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent

Method	Visibility	Documented in
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

Create

Creates and initializes a *TToolBar* instance.

constructor Create(AOwner: TComponent); **override**;

Description *Create* generates a *TToolBar* object and initializes its properties.

Destroy

Destroys the *TToolBar* instance and frees its memory

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the control is not already destroyed, and only then calls *Destroy*.

See also *Free* method

TToolButton

Unit: ComCtrls

TObject • TPersistent • TComponent • TControl • TGraphicControl • TToolButton

TToolButton is a button control in a *TToolBar* object.

Description Use *TToolButton* to implement buttons in a tool bar. While other controls (including *TButton* and *TSpeedButton*) can be placed in tool bars, *TToolButton* utilizes special tool-bar features to simplify the configuration of buttons and offers added display options such as pop-up borders and transparency.

To place tool buttons on a tool bar at design time, select the tool bar, right-click, and choose New Button.

See also *TButton* object, *TSpeedButton* object, *TToolBar* object, *TToolBar.Buttons* property

Properties

The following table lists the properties of *TToolButton*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
AllowAllUp	published		TToolButton
BoundsRect	public		TControl
Caption	published		TControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TControl
ClientRect	public, read-only		TControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
Down	published	False	TToolButton
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TControl
DropDownMenu	published		TToolButton

Property	Visibility	Default	Documented in
Enabled	published	True	TControl
Grouped	published	False	TToolButton
Height	published		TControl
Hint	published		TControl
ImageIndex	published		TToolButton
Indeterminate	published	False	TToolButton
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentShowHint	published	True	TControl
PopupMenu	published		TControl
ShowHint	published		TControl
Style	published	tbsButton	TToolButton
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl
Wrap	published	False	TToolButton

AllowAllUp

Allows all the tool buttons in a group to be unselected at the same time.

property AllowAllUp: Boolean;

Description If *AllowAllUp* is set to *True*, all of the tool buttons in the group can be unselected. If *AllowAllUp* is set to *False*, at least one tool button in the group must be selected at any time. (Determine which tool button is initially selected by setting its *Down* property to *True*.)

AllowAllUp is effective only when *Grouped* is *True* and *Style* is set to *tbsCheck*. Changing the value of *AllowAllUp* for one button in a group automatically changes the value for all buttons in the group.

See also *Down* property, *Grouped* property, *Style* property, *TSpeedButton.AllowAllUp* property

Down

Specifies whether the button is selected (down) or unselected (up).

property Down: Boolean;

Description Read *Down* to determine whether a tool button is selected. Setting *Down* to *True* at design time causes the button to be initially selected.

See also *AllowAllUp* property, *Grouped* property, *Indeterminate* property, *Style* property

DropdownMenu

Identifies a pop-up menu associated with the tool button.

property DropdownMenu: TPopupMenu;

Description Assign a value to *DropdownMenu* to make a menu appear when the button is pressed. (To display the menu programmatically, call the tool button's *Click* method.) If the menu's *AutoPopup* property is set to *True*, the menu appears automatically. If *AutoPopup* is *False*, display the menu by calling the menu's *Popup* method in the tool button's *OnClick* event handler.

See also *Click* method, *Down* property, *OnClick* event, *TPopupMenu* object, *TPopupMenu.AutoPupup* property, *TPopupMenu.Pupup* method

Grouped

Aggregates a sequence of adjacent *tbsCheck* buttons.

property Grouped: Boolean;

Description When an unbroken sequence of adjacent tool buttons each has its *Grouped* property set to *True* and *Style* set to *tbsCheck*, then no more than one of the buttons can be selected at the same time. Selecting a grouped button at runtime unselects the other buttons in its group.

To create distinct groups of buttons within a tool bar, separate the groups with any of the following: tool buttons whose *Grouped* property is *False*, tool buttons whose *Style* property is not *tbsCheck* (*tbsSeparator* and *tbsDivider* buttons are useful for this), or controls other than tool buttons.

See also *AllowAllUp* property, *Down* property, *Indeterminate* property, *Style* property

ImageIndex

Determines which images appears on the tool button.

property ImageIndex: Integer;

Description The *ImageIndex* property determines which image—from *TToolBar*'s *Images* list—appears on the tool button. If *TToolBar.DisabledImages* and *TToolBar.HotImages* are defined, different images appear when the button is disabled and when the mouse pointer passes over the button.

These lists are zero-based; the first image is indexed by 0, the second by 1, and so forth. Since *ImageIndex* holds only a single integer value, the positions

of the selected images must coincide on the different lists. For example, if *ImageIndex* = 2, the button displays the third image in each list.

See also *TToolBar.DisabledImages* property, *TToolBar.HotImages* property, *TToolBar.Images* property

Indeterminate

Indicates that the tool button is neither selected nor unselected.

```
property Indeterminate: Boolean;
```

Description *Indeterminate* provides a third state for tool buttons that are neither selected nor unselected. Set this property in program code to indicate that a button is inapplicable in a particular context. Buttons with the *Indeterminate* property set to *True* appear grayed.

When *Indeterminate* is set to *True*, *Down* is automatically set to *False*.

See also *Down* property

Style

Determines the style of the tool button.

```
type TToolButtonStyle = (tbsButton, tbsCheck, tbsDropDown, tbsSeparator, tbsDivider);
property Style: TToolButtonStyle;
```

Description These are the possible values of *Style*:

Value	Meaning
tbsButton	The button appears and functions like a normal speed button.
tbsCheck	Clicking the button toggles the *Down* property. Once selected, the button remains selected until clicked again.
tbsDropDown	The button displays a downward-pointing arrow (suitable for accessing a drop-down menu).
tbsSeparator	The button appears as an empty space on the tool bar (used to separate other controls).
tbsDivider	The button appears as a vertical line on the tool bar (used to separate other controls).

See also *AllowAllUp* property, *DropDownMenu*, *TSpeedButton* object

Wrap

Forces a new row after the tool button.

```
property Wrap: Boolean;
```

Description Set *Wrap* to *True* to make the tool button the last in its row.

See also *TToolBar.Wrapable* property

Events

The following table lists the events of *TToolButton*:

Event	Visibility	Documented in
OnClick	published	TControl
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods of *TToolButton*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Click	public	TControl
ClientToScreen	public	TControl
Create	public	TToolButton
DefaultHandler	public	TObject
Destroy	public	TToolButton
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent

Method	Visibility	Documented in
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTextBuf	public	TControl
GetTextLen	public	TControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
Invalidate	public	TControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
Perform	public	TControl
Refresh	public	TControl
RemoveComponent	public	TComponent
Repaint	public	TControl
SafeCallException	public	TComponent
ScreenToClient	public	TControl
SendToBack	public	TControl
SetBounds	public	TControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TControl

T

Create

Creates and initializes a *TToolButton* instance.

constructor Create(AOwner: TComponent); **override**;

Description Create generates a *TToolButton* and initializes its properties.

Destroy

Destroys the *TToolButton* and frees its memory.

destructor Destroy; **override**;

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the control is not already destroyed, and only then calls *Destroy*.

See also *Free* method

TToolWindow

Unit: ToolWin

TObject • TPersistent • TComponent • TControl • TWinControl • TToolWindow

TToolWindow is the abstract ancestor for windowed controls that require a custom client area and edge borders.

Description *TToolWindow* is the ancestor of the *TCoolBar* and *TToolBar* components, which facilitate the arrangement of buttons and other visual controls.

See Also *TCoolBar* object, *TToolBar* object

Properties

The following table lists the properties of *TToolWindow*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
BorderWidth	public	0	TToolWindow
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl

Property	Visibility	Default	Documented in
ControlStyle	public		TControl
Ctl3D	public		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
EdgeBorders	public	[ebLeft, ebTop, ebRight, ebBottom]	TToolWindow
EdgeInner	public	esRaised	TToolWindow
EdgeOuter	public	esLowered	TToolWindow
Enabled	public	True	TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentWindow	public		TWinControl
ShowHint	public		TControl
Showing	public, read-only		TWinControl
TabOrder	public	–1	TWinControl
TabStop	public	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	public	True	TControl
Width	published		TControl
WindowProc	public		TControl

BorderWidth

Determines the width of the control's internal border.

property BorderWidth: Integer;

Description The *BorderWidth* property creates a margin between the inner edge of the *TToolWindow* and the controls inside it. If *BorderWidth* is too large, the internal controls will not be visible.

EdgeBorders

Determines which sides of the control have visible borders.

```
type
  TEdgeBorder = (ebLeft, ebTop, ebRight, ebBottom);
  TEdgeBorders = set of TEdgeBorder;

property EdgeBorders: TEdgeBorders;
```

Description Use the *EdgeBorders* property to add or remove borders. The appearance of the borders can be set with the *EdgeInner* and *EdgeOuter* properties.

See also *EdgeInner* property, *EdgeOuter* property

EdgeInner

Determines the border style for the control's inner edge.

```
type TEdgeStyle = (esNone, esRaised, esLowered);

property EdgeInner: TEdgeStyle;
```

Description The possible values of *EdgeInner* are:

Value	Meaning
esNone	No inner edge is displayed.
esRaised	The inner edge is raised.
esLowered	The inner edge is lowered.

Use *EdgeInner* with *EdgeOuter* to give the component a three-dimensional look.

See also *EdgeBorders* property, *EdgeOuter* property

EdgeOuter

Determines the border style for the control's outer edge.

```
type TEdgeStyle = (esNone, esRaised, esLowered);

property EdgeOuter: TEdgeStyle;
```

Description The possible values of *EdgeOuter* are:

Value	Meaning
esNone	No outer edge is displayed.
esRaised	The outer edge is raised.
esLowered	The outer edge is lowered.

Use *EdgeOuter* with *EdgeInner* to give the component a three-dimensional look.

See also *EdgeBorders* property, *EdgeInner* property

Methods

The following table lists the methods of *TToolWindow*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TToolWindow
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TWinControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent

Method	Visibility	Documented in
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

Create

Creates and initializes a *TToolWindow* instance.

constructor Create(AOwner: TComponent); **override**;

Description Create generates a *TToolWindow* and initializes its properties.

TTrackBar

Unit:ComCtrls | Palette: Win32

TObject • TPersistent • TComponent • TControl • TWinControl • TTrackBar

TTrackBar is a wrapper for the Windows track bar control.

Description Use TTrackBar to put a track bar on a form. A track bar represents a position along a continuum using a slider and, optionally, tick marks. A track bar can also display a selected range marked by triangular ticks at the starting and ending positions of the selection.

During program execution, the slider can be moved to the desired position by dragging it with the mouse or by clicking the mouse on the bar. To use the keyboard to move the slider, press the arrow keys or the *PageUp* and *PageDown* keys.

See also *TScrollBar* object

Properties

The following table lists the properties of *TTrackBar*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl

Property	Visibility	Default	Documented in
DragMode	published	dmManual	TControl
Enabled	published	True	TControl
Frequency	published		TTrackBar
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Left	published		TControl
LineSize	published	1	TTrackBar
Max	published	10	TTrackBar
Min	published	0	TTrackBar
Name	published		TControl
Orientation	published		TTrackBar
Owner	public, read-only		TComponent
PageSize	published	2	TTrackBar
Parent	public		TControl
ParentCtl3D	published	True	TWinControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
Position	published		TTrackBar
SelEnd	published		TTrackBar
SelStart	published		TTrackBar
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent
TickMarks	published		TTrackBar
TickStyle	published		TTrackBar
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Frequency

Frequency specifies the increment between tick marks on the track bar.

property Frequency: Integer;

Description Use *Frequency* to specify the spacing of the tick marks, using the logical units used by the Position property. For example, a *Frequency* of 5 sets a tick mark at every fifth possible increment. To provide visual feedback about the units used by the track bar, set *Frequency* to *LineSize* or *PageSize*.

Note If the value of the *TickStyle* property is not set to *tsAuto*, the value of *Frequency* is ignored.

See also *LineSize* property, *PageSize* property, *SetTick* method, *TickStyle* property

LineSize

LineSize is the amount the value of *Position* changes when the user presses the arrow keys.

```
property LineSize: Integer;
```

Description Use *LineSize* to specify the distance along the slider that represents a single unit. The current *Position* is marked by a slider. The right and down arrow keys increment the value of *Position* by *LineSize*, and the left and up arrow keys decrement the value of *Position* by *LineSize*.

Use the *PageSize* property to set the number of ticks moved using the *PageUp* and *PageDown* keys.

See also *PageSize* property, *Position* property

Max

Max specifies the maximum *Position* of a *TTrackBar*.

```
property Max: Integer;
```

Description Use *Max* to set an upper limit to the value that can be represented using the track bar. A slider indicates the current *Position* in a range between *Min* and *Max*.

See also *Min* property, *Position* property

Min

Min specifies the minimum *Position* of a *TTrackBar*.

```
property Min: Integer;
```

Description Use *Min* to set a lower limit to the value that can be represented using the track bar. A slider indicates the current *Position* in a range between *Min* and *Max*.

See also *Max* property, *Position* property

Orientation

Orientation specifies whether the track bar is horizontal or vertical.

```
type TTrackBarOrientation = (trHorizontal, trVertical);
property Orientation: TTrackBarOrientation;
```

Description Set *Orientation* to the desired orientation of the track bar. *Orientation* has these possible values:

Value	Meaning
tbHorizontal	The long edge of the track is parallel to the top of the form. The *Min* position is on the left and the *Max* position is on the right.
tbVertical	The long edge of the track is parallel to the side of the form. The *Min* position is on the top and the *Max* position is on the bottom.

See also *BoundsRect* property

PageSize

PageSize specifies the amount *Position* is changed when the *PageUp* or *PageDown* key is pressed, or when the bar is clicked.

```
property PageSize: Integer;
```

Description Set *PageSize* to the distance along the track bar that corresponds to a single view. The current *Position* is marked by a slider. The *PageUp* key increments the value of *Position* by *PageSize*, and *PageDown* key decrements the value of *Position* by *PageSize*.

Use the *LineSize* property to set the number of ticks moved using the arrow keys.

See also *LineSize* property, *Position* property

Position

Position contains the current position of the slider of a *TTrackBar*.

```
property Position: Integer;
```

Description Read *Position* to determine the current value represented by the track bar. *Position* is a value in the range between *Min* and *Max* (inclusive). Set *Position* to programmatically move the slider of the track bar to a new value.

See also *Max* property, *Min* property

SelEnd

SelEnd specifies the position of the end point of the selection range.

```
property SelEnd: Integer;
```

Description Set *SelEnd* to determine the length of a selection that begins at *SelStart*. The track bar indicates a selection range on the bar by placing special tick marks at the positions specified by the *SelStart* and *SelEnd* properties and high-lighting the selected portion of the bar.

SelEnd must be greater than or equal to *Position*. If *SelEnd* is set to a value less than *Position*, *Position* changes to *SelEnd*. *SelEnd* must be greater than or equal to *SelStart*. If *SelEnd* is the same as *SelStart*, there is no selection, and the value of *Position* is the same as *SelStart* and *SelEnd*.

See also *Position* property, *SelStart* property

SelStart

SelStart specifies the position of the starting point of the selection range.

```
property SelStart: Integer;
```

Description Set *SelStart* to determine the beginning of a selection that runs to *SelEnd*. The track bar indicates a selection range on the bar by placing special tick marks at the positions specified by the *SelStart* and *SelEnd* properties and high-lighting the selected portion of the bar.

SelStart must be less than or equal to *Position*. If *SelStart* is set to a value greater than *Position*, *Position* changes to *SelStart*. *SelStart* must be less than or equal to *SelEnd*. If *SelStart* is the same as *SelEnd*, there is no selection, and the value of *Position* is the same as *SelStart* and *SelEnd*.

See also *Position* property, *SelEnd* property

TickMarks

The *TickMarks* property specifies the location of the tick marks.

```
type TTickMark = (tmBottomRight, tmTopLeft, tmBoth);
```

```
property TickMarks: TTickMark;
```

Description Set *TickMarks* to specify where the tick marks should be drawn. *TickMarks* has these possible values:

Value	Meaning
tmBottomRight	Tick marks are on the bottom or right of the track bar, depending of the value of *Orientation*.
tmTopLeft	Tick marks are on the top or left of the track bar, depending of the value of *Orientation*.
tmBoth	Tick marks are on both sides of the track bar.

If the *TickStyle* property is *tsNone*, the value of *TickMarks* is ignored.

See also *TickStyle* property

T

TickStyle

TickStyle specifies how tick marks are placed on the track bar.

```
property TickStyle: TTickStyle;
```

Description Set *TickStyle* to specify whether the track bar should display tick marks, and if so, how those tick marks are set. *TickStyle* has these possible values:

Value	Meaning
tsAuto	Tick marks are automatically displayed at increments equal to the value of the *Frequency* property.
tsManual	Tick marks are displayed at the *Min* and *Max* values. Additional tick marks can be set using the *SetTick* method.
tsNone	No tick marks are displayed.

See also *Frequency* property, *Max* property, *Min* property, *SetTick* method

Events

The following table lists the events of *TTrackBar*:

Event	Visibility	Documented in
OnChange	published	TTrackBar
OnDragDrop	published	TControl
OnDragOver	published	TControl
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnStartDrag	published	TControl

OnChange

OnChange occurs immediately after the value of *Position* changes.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnChange: TNotifyEvent;
```

Description Write an *OnChange* event handler to take specific action whenever the position of the slider may have changed. For example, if the track bar is being used to control another object, update the other object from an *OnChange* event handler.

See also *Position* property

Methods

The following table lists the methods of *TTrackBar*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TTrackBar
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TWinControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent

Method	Visibility	Documented in
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
SetTick	public	TTrackBar
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

Create

Create creates and initializes an instance of *TTrackBar*.

constructor Create(AOwner: TComponent); **override**;

Description Call *Create* to instantiate a *TTrackBar* object at runtime. Track bars placed on forms at design time are created automatically.

Create allocates memory and initializes the following properties:

- *ControlStyle* is set to [*csCaptureMouse, csClickEvents, csSetCaption*].
- *Frequency* is set to 1.
- *Height* is set to 45 and *Width* to 150.
- *LineSize* is set to 1 and *PageSize* to 2.
- *Max* is set to 10 and *Min* to 0.
- *Orientation* is set to *trHorizontal*.
- *TickMarks* is set to *tbBottomRight* and *TickStyle* to *tsAuto*.
- *TabStop* is set to *True*.

See also *ControlStyle* property, *Frequency* property, *Height* property, *LineSize* property, *Max* property, *Min* property, *Orientation* property, *PageSize* property, *TabStop* property, *TickMarks* property, *TickStyle* property, *TWinControl.Create* method

SetTick

SetTick adds a tick mark to the track bar at a specified location.

```
procedure SetTick(Value: Integer);
```

Description Use *SetTick* to place a tick mark on the bar at the position that corresponds to the *Value* parameter. Tick marks specified in this manner are only displayed when the *TickStyle* is set to *tsManual*.

See also *Position* property, *TickStyle* property

TTreeView

Unit: ComCtrls — Pallete: Win32

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomTreeView • TTreeView

A tree view control is a window that displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk.

Description Each node in a tree view control consists of a label and a number of optional bitmapped images. Each node can have a list of subnodes associated with it. By clicking on a node, the user can expand and collapse the associated list of subnodes. This control can be seen as the left pane of the *Database Explorer*.

See also *TTreeNode* object, *TTreeNodes* object

Properties

The following table lists the properties of *TTreeView*:

Property	Visibility	Default	Documented in
Align	published	alNone	TControl
BorderStyle	published	bsSingle	TCustomTreeView
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
Color	published	clWindow	TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	published		TWinControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
DragCursor	published	crDrag	TControl
DragMode	published	dmManual	TCustomTreeView
DropTarget	public		TCustomTreeView
Enabled	published	True	TControl
Font	published		TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
HideSelection	published	True	TCustomTreeView
Hint	published		TControl
Images	published		TCustomTreeView
Indent	published		TCustomTreeView
Items	published		TCustomTreeView
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent

Property	Visibility	Default	Documented in
Parent	public		TControl
ParentColor	published	False	TControl
ParentCtl3D	published	True	TWinControl
ParentFont	published	True	TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
ReadOnly	published	False	TCustomTreeView
Selected	public		TCustomTreeView
ShowButtons	published	True	TCustomTreeView
ShowHint	published		TControl
Showing	public, read-only		TWinControl
ShowLines	published	True	TCustomTreeView
ShowRoot	published	True	TCustomTreeView
SortType	published	stNone	TCustomTreeView
StateImages	published		TCustomTreeView
TabOrder	published	–1	TWinControl
TabStop	published	True	TWinControl
Tag	published	0	TComponent
Top	published		TControl
TopItem	public		TCustomTreeView
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl

Events

The following table lists the events for *TTreeView*:

Event	Visibility	Documented in
OnChange	published	TCustomTreeView
OnChanging	published	TCustomTreeView
OnClick	published	TControl
OnCollapsed	published	TCustomTreeView
OnCollapsing	published	TCustomTreeView
OnCompare	published	TCustomTreeView
OnDblClick	published	TControl
OnDeletion	published	TCustomTreeView
OnDragDrop	published	TControl
OnDragOver	published	TControl

Event	Visibility	Documented in
OnEdited	published	TCustomTreeView
OnEditing	published	TCustomTreeView
OnEndDrag	published	TControl
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnExpanded	published	TCustomTreeView
OnExpanding	published	TCustomTreeView
OnGetImageIndex	published	TCustomTreeView
OnGetSelectedIndex	published	TCustomTreeView
OnKeyDown	published	TWinControl
OnKeyPress	published	TWinControl
OnKeyUp	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl
OnStartDrag	published	TControl

Methods

The following table lists the methods for *TTreeView*:

Method	Visibility	Documented in
AlphaSort	public	TCustomTreeView
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomTreeView
CreateParented	public	TWinControl
CustomSort	public	TCustomTreeView
DefaultHandler	public	TObject

Method	Visibility	Documented in
Destroy	public	TCustomTreeView
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
FullCollapse	public	TCustomTreeView
FullExpand	public	TCustomTreeView
GetHitTestInfoAt	public	TCustomTreeView
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetNodeAt	public	TCustomTreeView
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
IsEditing	public	TCustomTreeView
LoadFromFile	public	TCustomTreeView
LoadFromStream	public	TCustomTreeView
MethodAddress	public	TObject

Method	Visibility	Documented in
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
SaveToFile	public	TCustomTreeView
SaveToStream	public	TCustomTreeView
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

TUpdateSQL

Unit: DBTables | Palette: Data Access

TObject • TPersistent • TComponent • TDataSetUpdateObject • TUpdateSQL

Provides an object for updating read-only datasets when cached updates are enabled.

Description

Use a *TUpdateSQL* object to provide SQL statements used to update read-only datasets represented by *TQuery* components when cached updates are enabled. A dataset is read-only either by design or circumstance. If a dataset is read-only by design, the application itself does not provide a user interface for updating data, but may institute a programmatic scheme behind the scenes. If a dataset is read-only by circumstance, it indicates that the Borland Database Engine (BDE) returned a read-only result set. This usually happens for queries made against multiple tables. Such queries are, by SQL-92 definitions, read-only.

TUpdateSQL provides a mechanism for circumventing what some developers consider an SQL-92 limitation. It enables a developer to provide INSERT,

UPDATE, and DELETE statements for performing separate update queries that are transparent to the end user.

In practical application, a *TUpdateSQL* object is placed on a data module or form, and linked to a *TQuery* component through that component's *UpdateSQL* property. If the *UpdateSQL* property points to a valid *TUpdateSQL* object, the SQL statements belonging to the update object are automatically applied when cached updates are applied.

See also *TQuery* object, *TQuery.UpdateSQL* property

Properties

The following table lists the properties of *TUpdateSQL*:

Property	Visibility	Default	Documented in
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DataSet	public		TUpdateSQL
DeleteSQL	published		TUpdateSQL
DesignInfo	public		TComponent
InsertSQL	published		TUpdateSQL
ModifySQL	published		TUpdateSQL
Name	published		TComponent
Owner	public, read-only		TComponent
Query	public, read-only		TUpdateSQL
SQL	public		TUpdateSQL
Tag	published	0	TComponent
VCLComObject	public		TComponent

DataSet

Identifies the dataset to which a *TUpdateSQL* component belongs.

property DataSet: TDataSet;

Description At design time *DataSet* is automatically set to identify the dataset object that uses it when setting the dataset object's *UpdateSQL* property to the name of the update object. An application should only need to set this property if it creates a new update component at runtime.

See also *TQuery.UpdateSQL* property

DeleteSQL

Specifies the SQL DELETE statement to use when applying a cached deletion of a record.

property DeleteSQL: TStrings;

Description Set *DeleteSQL* to the SQL DELETE statement to use when applying a deletion to a record. Statements can be parameterized queries. To create a DELETE statement at design time, use the UpdateSQL editor to create statements, such as"

```
delete from "COUNTRY.DB"
where
  Name = :OLD_Name
```

At runtime, an application can write a statement directly to this property to set or change the DELETE statement.

Note As the example illustrates, *DeleteSQL* supports an extension to normal parameter binding. To retrieve the value of a field as it exists prior to application of cached updates, prepend the field name with 'OLD_'. This is especially useful when doing field comparisons in the WHERE clause of the statement.

See also *InsertSQL* property, *ModifySQL* property

InsertSQL

Specifies the SQL INSERT statement to use when applying a cached insertion of a record.

property InsertSQL: TStrings;

Description Set *InsertSQL* to the SQL INSERT statement to use when applying an insertion to a dataset. Statements can be parameterized queries. To create a INSERT statement at design time, use the UpdateSQL editor to create statements, such as"

```
insert into "COUNTRY.DB"
  (Name, Capital, Continent)
  value (:Name, :Capital, :Continent)
  where :OLD_Name = 'Rangoon'
```

At runtime, an application can write a statement directly to this property to set or change the INSERT statement.

Note As the example illustrates, *InsertSQL* supports an extension to normal parameter binding. To retrieve the value of a field as it exists prior to application of cached updates, prepend the field name with 'OLD_'. This is especially useful when doing field comparisons in the WHERE clause of the statement.

See also *DeleteSQL* property, *ModifySQL* property

ModifySQL

Specifies the SQL UPDATE statement to use when applying an update to a record and cached updates is enabled.

property ModifySQL: TStrings;

Description Set *ModifySQL* to the SQL UPDATE statement to use when applying an updated record to a dataset. Statements can be parameterized queries. To create a UPDATE statement at design time, use the UpdateSQL editor to create statements, such as"

```
update "COUNTRY.DB"
  set Name = :Name, Capital = :Capital, Continent = :Continent
  where Name = :OLD_Name
```

At runtime, an application can write a statement directly to this property to set or change the UPDATE statement.

Note As the example illustrates, *ModifySQL* supports an extension to normal parameter binding. To retrieve the value of a field as it exists prior to application of cached updates, prepend the field name with 'OLD_'. This is especially useful when doing field comparisons in the WHERE clause of the statement.

See also *DeleteSQL* property, *InsertSQL* property

Query

Returns the query object used to perform a specified kind of update.

type TUpdateKind = (ukModify, ukInsert, ukDelete);

property Query[UpdateKind: TUpdateKind]: TQuery;

Description Set *Query* to return the *TQuery* object used to perform a particular form of SQL update. *UpdateKind* specifies which query object to retrieve. *UpdateKind* can be one of the following:

Value	Meaning
ukModify	Return the query object used to execute UPDATE statements.
ukInsert	Return the query object used to execute INSERT statements.
ukDelete	Return the query object used to execute DELETE statements

Each query object executes a particular kind of SQL statement. The contents of the SQL statements executed by these objects can be accessed directly using the *ModifySQL*, *InsertSQL*, and *DeleteSQL* properties.

The main purpose of *Query* is to provide a a way for an application to set the properties for an update query object or to call the query object's methods.

Note If a particular kind of update statement is not provided, then its corresponding query object is **nil**. For example, if an application does not

provide an SQL statement for the *DeleteSQL* property, then setting *Query[ukDelete]* returns **nil**.

See also *DeleteSQL* property, *InsertSQL* property, *ModifySQL* property

SQL

Returns a specified SQL statement used when applying cached updates.

```
type TUpdateKind = (ukModify, ukInsert, ukDelete);
property SQL[UpdateKind: TUpdateKind]: TStrings;
```

Description Returns the SQL statement in the *ModifySQL*, *InsertSQL*, or *DeleteSQL* property, depending on the setting of *UpdateKind*. *UpdateKind* can be any of the following:

Value	Meaning
ukModify	Return the SQL statement used to update records in the dataset
ukInsert	Return the SQL statement used to insert new records into the dataset
ukDelete	Return the SQL statement used to delete records in the dataset

See also *DeleteSQL* property, *InsertSQL* property, *ModifySQL* property

Methods

The following table lists the methods of *TUpdateSQL*:

Method	Visibility	Documented in
Apply	public	TUpdateSQL
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TUpdateSQL
DefaultHandler	public	TObject
Destroy	public	TUpdateSQL
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
ExecSQL	public	TUpdateSQL
FieldAddress	public	TObject
FindComponent	public	TComponent

Method	Visibility	Documented in
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent
SetParams	public	TUpdateSQL

Apply

Sets the parameters for a specified SQL statement type, and executes the resulting statement.

```
type TUpdateKind = (ukModify, ukInsert, ukDelete);
procedure Apply(UpdateKind: TUpdateKind);
```

Description Call *Apply* to set parameters for an SQL statement and execute it to update a record. *UpdateKind* indicates which SQL statement to bind and execute, and can be one of the following values:

Value	Meaning
ukModify	Bind and execute the SQL statement in the *ModifySQL* property
ukInsert	Bind and execute the SQL statement in the *InsertSQL* property
ukDelete	Bind and execute the SQL statement in the *DeleteSQL* property

Apply is primarily intended for manually executing update statements from an *OnUpdateRecord* event handler.

Note If an SQL statement does not contain parameters, it is more efficient to call *ExecSQL* instead of *Apply*.

See also *DeleteSQL* property, *ExecSQL* method, *InsertSQL* property, *ModifySQL* property, *SetParams* method

ExecSQL

Executes a specified type of SQL statement to perform an update for an otherwise read-only results set when cached updates is enabled.

```
type TUpdateKind = (ukModify, ukInsert, ukDelete);
procedure ExecSQL(UpdateKind: TUpdateKind);
```

Description Call *ExecSQL* to execute the SQL statement necessary for updating the records belonging to a read-only result set when cached updates is enabled. *UpdateKind* specifies the statement to execute, and can be one of the following values:

Value	Meaning
ukModify	Execute the SQL statement used to update records in the dataset
ukInsert	Execute the SQL statement used to insert new records into the dataset
ukDelete	Execute the SQL statement used to delete records in the dataset

If the statement to execute contains any parameters, an application must call *SetParams* to bind the parameters before calling *ExecSQL*. To determine if a statement contains parameters, examine the appropriate *ModifySQL*, *InsertSQL*, or *DeleteSQL* property, depending on the statement type intended for execution.

Note To both bind parameters and execute a statement, call *Apply*.

See also *Apply* method, *DeleteSQL* property, *InsertSQL* property, *ModifySQL* property, *SetParams* method

SetParams

Binds parameters in an SQL statement prior to statement execution.

```
type TUpdateKind = (ukModify, ukInsert, ukDelete);
procedure SetParams(UpdateKind: TUpdateKind);
```

Description Call *SetParams* to bind parameters in an SQL statement associated with the update object prior to executing the statement. *UpdateKind* indicates the type of statement for which to bind parameters, and can be one of the following values:

Value	Meaning
ukModify	Bind parameters for SQL statement used to update records
ukInsert	Bind parameters for the SQL statement used to insert new records
ukDelete	Bind parameters for the SQL statement used to delete records

Parameters are indicated in an SQL statement by a colon. Except for the leading colon in the parameter name, the parameter name must exactly match the name of an existing field name for the dataset.

Note Parameter names can be prefaced by the 'OLD_' indicator. If so, the old value of the field is used to perform the update instead of any updates in the cache.

See also *DeleteSQL* property, *ExecSQL* method, *InsertSQL* property, *ModifySQL* property

TUpDown

Unit: ComCtrls | Palette: Win32

TObject • TPersistent • TComponent • TControl • TWinControl • TCustomUpDown • TUpDown

TUpDown is a wrapper for a Windows up-down control.

Description Use *TUpDown* to add an up-down control to a form. Up-down controls consist of a pair of arrow buttons, such as the arrows that appear in a spin box. Up-down controls allow users to change the size of a numerical value by clicking on arrow buttons.

See also *TSpinButton* object, *TTrackBar* object

Properties

The following table lists the properties for *TUpDown*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
AlignButton	published	udRight	TCustomUpDown
ArrowKeys	published	True	TCustomUpDown
Associate	published		TCustomUpDown
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent

U

Property	Visibility	Default	Documented in
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Cursor	published	crDefault	TControl
DesignInfo	public		TComponent
Enabled	published	True	TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl
Increment	published	1	TCustomUpDown
Left	published		TControl
Max	published	100	TCustomUpDown
Min	published		TCustomUpDown
Name	published		TControl
Orientation	published	udVertical	TCustomUpDown
Owner	public, read-only		TComponent
Parent	public		TControl
ParentShowHint	published	True	TControl
ParentWindow	public		TWinControl
PopupMenu	published		TControl
Position	published		TCustomUpDown
ShowHint	published		TControl
Showing	public, read-only		TWinControl
TabOrder	published	–1	TWinControl
TabStop	published	False	TWinControl
Tag	published	0	TComponent
Thousands	published	True	TCustomUpDown
Top	published		TControl
VCLComObject	public		TComponent
Visible	published	True	TControl
Width	published		TControl
WindowProc	public		TControl
Wrap	published		TCustomUpDown

Events

The following table lists the events for *TUpDown*:

Event	Visibility	Documented in
OnChanging	published	TCustomUpDown
OnClick	published	TCustomUpDown
OnEnter	published	TWinControl
OnExit	published	TWinControl
OnMouseDown	published	TControl
OnMouseMove	published	TControl
OnMouseUp	published	TControl

Methods

The following table lists the methods for *TUpDown*:

Method	Visibility	Documented in
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TCustomUpDown
CreateParented	public	TWinControl
DefaultHandler	public	TObject
Destroy	public	TWinControl
DestroyComponents	public	TComponent
Destroying	public	TComponent
DisableAlign	public	TWinControl
Dispatch	public	TObject
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl

Method	Visibility	Documented in
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
HandleAllocated	public	TWinControl
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
PaintTo	public	TWinControl
Perform	public	TControl
Realign	public	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl

Method	Visibility	Documented in
SetFocus	public	TWinControl
SetTextBuf	public	TControl
Show	public	TControl
Update	public	TWinControl
UpdateControlState	public	TWinControl

TVarBytesField

Unit: DB

TObject • TPersistent • TComponent • TField • TBinaryField • TBytesField • TVarBytesField

A *TVarBytesField* object represents a variable-length untyped binary field in a dataset.

Description A variable bytes field value is a set of bytes of variable size, with the actual length of the value stored in the first two bytes.

TVarBytesField differs from its immediate ancestor *TBytesField* only in the data type it represents. Except for the constructor and the *DataSize* property, all of the properties, methods, and events are the same as those of *TBytesField*, and provide functionality that is useful for managing unformatted binary data fields in a dataset.

See also *TDataSet object*

Properties

The following table lists the properties of *TVarBytesField*:

Property	Visibility	Default	Documented in
Alignment	published	taLeftJustify	TField
AsBoolean	public		TField
AsCurrency	public		TField
AsDateTime	public		TField
AsFloat	public		TField
AsInteger	public		TField
AsString	public		TBinaryField
AsVariant	public		TBinaryField
AttributeSet	public		TField
Calculated	public	False	TField
CanModify	public, read-only		TField
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent

V

Property	Visibility	Default	Documented in
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ConstraintErrorMessage	published		TField
CurValue	public, read-only		TField
CustomConstraint	published		TField
DataSet	public		TField
DataSize	public, read-only		TVarBytesField
DataType	public, read-only		TField
DefaultExpression	published		TField
DesignInfo	public		TComponent
DisplayLabel	published		TField
DisplayName	public, read-only		TField
DisplayText	public, read-only		TField
DisplayWidth	published		TField
EditMask	public		TField
EditMaskPtr	public, read-only		TField
FieldKind	published	fkData	TField
FieldName	published		TField
FieldNo	public, read-only		TField
ImportedConstraint	published		TField
Index	published		TField
IsIndexField	public, read-only		TField
IsNull	public, read-only		TField
KeyFields	published		TField
Lookup	public		TField
LookupCache	published	False	TField
LookupDataSet	published		TField
LookupKeyFields	published		TField
LookupList	public, read-only		TField
LookupResultField	published		TField
Name	published		TComponent
NewValue	public		TField
Offset	public, read-only		TField
OldValue	public, read-only		TField
Origin	published		TField
Owner	public, read-only		TComponent
ReadOnly	published	False	TField
Required	published	False	TField
Size	published	16	TField
Tag	published	0	TComponent
Text	public		TField

Property	Visibility	Default	Documented in
ValidChars	public		TField
Value	public		TBinaryField
VCLComObject	public		TComponent
Visible	published	True	TField

DataSize

DataSize is the amount of memory that is needed to store the value of a variable bytes field in its native database format.

```
property DataSize: Word;
```

Description Check *DataSize* to determine the number of bytes required to store a variable bytes field component's value. Use *DataSize* to determine the size needed for a buffer when working with the *GetData* and *SetData* methods.

For variable bytes fields, *DataSize* is *Size* + 2. The unformatted data can take up to *Size* bytes, and the extra 2 bytes store the length of the unformatted data.

See also *GetData* method, *SetData* method, *Size* property

Events

The following table lists the events of *TVarBytesField*:

Event	Visibility	Documented in
OnChange	published	TField
OnGetText	published	TField
OnSetText	published	TField
OnValidate	published	TField

Methods

The following table lists the methods of *TVarBytesField*:

Method	Visibility	Documented in
Assign	public	TField
AssignValue	public	TField
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject

Method	Visibility	Documented in
Clear	public	TField
Create	public	TVarBytesField
DefaultHandler	public	TObject
Destroy	public	TField
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
FocusControl	public	TField
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetData	public	TField
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
IsBlob	public	TField
IsValidChar	public	TField
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RefreshLookupList	public	TField
RemoveComponent	public	TComponent
SafeCallException	public	TComponent
SetData	public	TField
SetFieldType	public	TField
Validate	public	TField

Create

The *Create* method creates an instance of *TVarBytesField*.

constructor Create(AOwner: TComponent); **override**;

Description It is seldom necessary to call *Create* directly, because a VarBytes field object is instantiated automatically for all variable bytes fields in a dataset.

After calling the inherited constructor, *Create* sets the *DataType* to *ftVarBytes* and initializes the *Size* property to 16.

See also *DataType* property, *Size* property, *TField.Create* method

TWebApplication

Unit: HTTPApp

TObject • TPersistent • TComponent • TWebApplication

TWebApplication encapsulates a Web server application.

Description *TWebApplication* provides the fundamental behavior of a Web server application. *TWebApplication* provides methods to create *TWebRequest* and *TWebResponse* objects that represent HTTP request messages and their responses, pass these objects to the dispatcher so that the response can be filled in, and send the response back to the Web client.

Each Web server project automatically declares an *Application* variable of type *TWebApplication* as the instance of the application. For CGI applications, this variable is initialized to a *TCGIApplication* object in the CGIApp unit. For ISAPI applications, this variable is initialized to a *TISAPIApplication* object in the ISAPIApp unit.

The *Application* variable is added to a project by selecting File | New from the menu of the Delphi main window and selecting Web Server Application in the New Items dialog. Do not add the Forms unit to the **uses** list of the project, because that unit declares a conflicting *Application* variable of type *TApplication*. In addition, do not add the ISAPIApp unit to a CGI application or the CGIApp unit to an ISAPI or NSAPI application. The CGIApp unit and the ISAPIApp unit each create the initial instance of the *Application* variable. Including both units in the **uses** list causes the memory for the object created in the first unit to be lost.

See also *Application* variable, *TCGIApplication* object, *TISAPIApplication* object, *TWebDispatcher* object, *TWebRequest* object, *TWebResponse* object

Properties

The following table lists the properties of *TWebApplication*:

Property	Visibility	Default	Documented in
ActiveCount	public, read-only		TWebApplication
CacheConnections	public		TWebApplication
ComObject	public, read-only		TComponent

W

Property	Visibility	Default	Documented in
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DesignInfo	public		TComponent
InactiveCount	public, read-only		TWebApplication
MaxConnections	public		TWebApplication
Name	published		TComponent
Owner	public, read-only		TComponent
Tag	published	0	TComponent
Title	public		TWebApplication
VCLComObject	public		TComponent

ActiveCount

ActiveCount is the number of active connections currently being serviced by the application.

```
property ActiveCount: Integer;
```

Description Use *ActiveCount* to monitor the number of HTTP request messages the application is servicing at any one time. Over time, this value provides an indication of the optimal value for the *MaxConnections* property.

Each time an ISAPI application receives an HTTP request message, it generates a new instance of the Web data module. After the response is returned to the Web client, the Web data module is either destroyed or deactivated and cached for later reuse. Each active Web data module resides in a separate execution thread.

ActiveCount is the number of Web data modules currently in use. *InactiveCount* is the number of deactivated Web data modules cached for later reuse. *MaxConnections* is the maximum number of Web data modules that can be active at any time. *ActiveCount* can never exceed *MaxConnections*.

Note *ActiveCount* is only meaningful for ISAPI applications. Each CGI application handles only a single request message.

See also *ActivateWebModule* method, *CacheConnections* property, *InactiveCount* property, *MaxConnections* property, *TWebModule* object

CacheConnections

CacheConnections determines whether the Web modules used by the application are cached for re-use.

```
property CacheConnections: Boolean;
```

Description Each time an ISAPI or NSAPI application receives an HTTP request message, it spawns a thread to handle the request and creates a new Web module that contains instances of all the objects used in the Web application. To improve performance, these Web modules can be saved in a cache when the thread associated with the HTTP request message finishes executing.

Set *CacheConnections* to *True* to have the Web application save inactive Web modules in a cache. Caching the Web modules speeds performance as new instances of the Web module and its contents do not need to be created every time a request message is received. Set *CacheConnections* to *False* to have the Web application create a new Web module for every request thread. Recreating the Web module ensures that changes made to the module or its contents when servicing the last request do not cause unexpected behavior the next time the module is activated.

Note *CacheConnections* is only relevant to ISAPI and NSAPI Web applications.

See also *InactiveCount* property, *TISAPIApplication* object, *TWebModule* object

InactiveCount

InactiveCount is the number of Web data modules cached for later reuse.

```
property InactiveCount: Integer;
```

Description Use *InactiveCount* to monitor the number of unused Web data modules. Over time, this value provides an indication of the optimal value for the *MaxConnections* property.

When *CacheConnections* is *True*, each time an ISAPI application receives an HTTP request message, it reactivates one of the inactive Web data modules. The application generates a new instance of the Web data module if *InactiveCount* is 0. After the response is returned to the Web client, the Web data module is deactivated again, and counted by the *InactiveCount* property.

When *CacheConnections* is *False*, a new Web data module is created for each HTTP request message and destroyed when the response is returned. *InactiveCount* is always 0 when *CacheConnections* is *False*.

Note *InactiveCount* is only meaningful for ISAPI applications. Each CGI application handles only a single request message, so there is no need to cache unused data modules.

See also *ActiveCount* property, *CacheConnections* property, *DeactivateWebModule* method, *MaxConnections* property, *TWebModule* object

MaxConnections

MaxConnections is the maximum number of request messages the application can handle at any one time.

```
property MaxConnections: Integer;
```

Description Set *MaxConnections* to adjust the number of HTTP request messages the application can handle simultaneously. Use the *ActiveCount* and *InactiveCount* properties to get a sense of the frequency and duration of request messages. Then set *MaxConnections* so that it is high enough to avoid exceptions when the application runs out of Web data modules, but low enough not to degrade performance too severely by running too many simultaneous threads.

If a request message is received when *ActiveCount* is already at *MaxConnections*, the application raises an exception.

Note *MaxConnections* is only meaningful for ISAPI applications. Each CGI application handles only a single request message.

See also *ActivateWebModule* method, *ActiveCount* property, *InactiveCount* property, *TWebModule* object

Title

Title contains the text that appears below the icon representing the application when it is minimized.

```
property Title: string;
```

Description Use the *Title* property to modify the name of this text string appearing below the icon of the application. The default value is taken from the DLL/EXE filename at runtime.

Do not confuse *Title* with the *Name* property. *Name* is the identifier for the Web application when it is referred to in code.

See also *Name* property

Methods

The following table lists the methods of *TWebApplication*:

Method	Visibility	Documented in
ActivateWebModule	protected	TWebApplication
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TWebApplication
CreateForm	public	TWebApplication
DeactivateWebModule	protected	TWebApplication

Method	Visibility	Documented in
DefaultHandler	public	TObject
Destroy	public	TWebApplication
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
DoHandleException	protected	TWebApplication
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HandleRequest	protected	TWebApplication
HasParent	public	TComponent
InheritsFrom	public	TObject
Initialize	public	TWebApplication
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
Run	public	TWebApplication
SafeCallException	public	TComponent

ActivateWebModule

ActivateWebModule activates a Web data module for the application and returns it.

```
function ActivateDataModule: TDataModule;
```

Description

ActivateDataModule is called internally when the application receives an HTTP request message. If the *InactiveCount* property is greater than zero, a cached Web data module is reactivated. Otherwise, a new instance of the Web data module is created.

Before *ActivateDataModule* is called, the *CreateForm* method must be called to register the data module.

A Web application object contains only a single Web data module. Usually, that Web data module is a *TWebModule* object, but it may also be an ordinary data module that contains a *TWebDispatcher* object. If there is no Web data module associated with the Web application, *ActivateDataModule* raises an exception. If the *ActiveCount* property is already at *MaxConnections* when *ActivateWebModule* is called, *ActivateWebModule* raises an exception.

See also *ActiveCount* property, *CreateForm* method, *InactiveCount* property, *MaxConnections* property, *TWebDispatcher* object, *TWebModule* object

Create

Create creates an instance of *TWebApplication*.

```
constructor Create(AOwner: TComponent); override;
```

Description Do not call *Create*. The single Web application object named *Application* is instantiated automatically when the *CGIApp* unit or the *ISAPIApp* unit is included in the uses list of the project.

Create allocates the helper objects used by *TWebApplication*.

See also *TComponent.Create* method

CreateForm

CreateForm registers the single Web data module with the Web application.

```
procedure CreateForm(InstanceClass: TComponentClass; var Reference); virtual;
```

Description Do not call *CreateForm* in an application. Delphi's module manager calls *CreateForm* internally to register the Web data module for the Web application.

Create registers the class of the Web data module for the application. The Web data module object is created later when the *ActivateDataModule* method is called for the first time.

See also *ActivateDataModule* object, *TWebModule* object

DeactivateWebModule

DeactivateWebModule deactivates the Web data module of the application.

```
procedure DeactivateDataModule(DataModule: TDataModule);
```

Description Do not call *DeactivateDataModule* directly. The Web application calls *DeactivateDataModule* internally after handling an HTTP request message. When *CacheConnections* is *True*, the Web data module remains inactive until the next request message is received. When *CacheConnections* is *False*, the runtime instance of the Web data module is destroyed.

See also *ActivateDataModule* method, *CacheConnections* property, *HandleRequest* method, *InactiveCount* property

Destroy

Destroy destroys an instance of *TWebApplication*.

```
destructor Destroy; override;
```

Description Never call the *Destroy* or *Free* method.The application is freed automatically when the application stops running.

Destroy frees the helper objects that were created at startup, as well as all Web data modules created by the application.

See also *ActivateWebModule* method, *Free* method, *Run* method

DoHandleException

DoHandleException provides the interface for a method that handles exceptions raised within the application.

```
procedure DoHandleException(E: Exception); dynamic;
```

Description The *DoHandleException* method for *TWebApplication* discards all exceptions. Descendants of *TWebApplication* can override this method to provide special responses when exceptions occur within the application.

See also *Exception* object

HandleRequest

HandleRequest responds to an HTTP request message received by the server.

```
function HandleRequest(Request: TWebRequest; Response: TWebResponse): Boolean;
```

Description When a descendant of *TWebApplication* receives an HTTP request message, it creates a *TWebRequest* object to represent the message, and a *TWebResponse* message to match the request. It then calls *HandleRequest*, passing those objects as the *Request* and *Response* parameters.

HandleRequest activates the Web data module associated with the Web application. It then passes the request and response objects to the dispatcher, so that it can dispatch the request to the appropriate action items. When the dispatcher returns, *HandleRequest* checks whether the response was handled but not sent, and if not, sends it. Finally, *HandleRequest* deactivates the data module.

HandleRequest returns a *Boolean* value indicating whether the request message was handled and sent.

See also *ActivateDataModule* method, *DeactivateDataModule* method, *TCustomWebDispatcher.DispatchAction* method, *TWebRequest* object, *TWebResponse* object

Initialize

Initialize introduces an interface for a method that initializes the application.

procedure Initialize; **virtual**;

Description The *Initialize* method for *TWebApplication* does nothing. *Initialize* is the first method called for every Delphi project. *Descendants* of *TWebApplication* that must perform any initialization before the application starts running override this method for that purpose.

Note Although *Initialize* is the first method called in the project source code, it is not the first code that is executed in an application.

See also *Run* method

Run

Run executes the Web application.

procedure Run; **virtual**;

Description When creating a new project, Delphi automatically creates a main program block in the project file that calls the *Run* method. When the application is executed, the application's *Run* method is called.

The *Run* method for *TWebApplication* sets up the default error handler and shut down sequence for the application. Descendants of *TWebApplication* override the *Run* method to receive HTTP request messages, create the *TWebRequest* and *TWebResponse* objects for incoming requests, and call the *HandleRequest* method to process those requests.

See also *HandleRequest* method, *TWebRequest* object, *TWebResponse* object

TWebDispatcher

Unit: HTTPApp | Palette: Internet

TObject • TPersistent • TComponent • TDataModule • TCustomWebDispatcher • TWebDispatcher

TWebDispatcher passes HTTP request message on to the appropriate action items that assemble a response.

Description Use *TWebDispatcher* to convert an ordinary data module to a Web module. Add a *TWebDispatcher* object to an existing data module that contains other non-visual components and business rules for the Web application. Then delete the automatically generated Web module from the Web application and replace it with the data module that now contains the Web dispatcher.

TWebDispatcher enables the Web server application to respond to HTTP request messages. Only one Web dispatcher can be added to an application. This is why the automatically generated Web module, which is a type of Web

dispatcher, must be deleted from the Web application before the data module containing *TWebDispatcher* can be added.

The *TWebDispatcher* object manages a collection of action items which know how to respond to HTTP request messages. View, add and delete the action items managed by the dispatcher using the Action editor which is opened by double clicking on the *TWebDispatcher* icon.

TWebDispatcher publishes some of the properties and events introduced by *TCustomWebDispatcher*, but introduces no new properties, events, or methods of its own.

Properties

The following table lists the properties of *TWebDispatcher*:

Property	Visibility	Default	Documented in
Action	public, read-only		TCustomWebDispatcher
Actions	published		TCustomWebDispatcher
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DesignInfo	public		TComponent
DesignOffset	public		TDataModule
DesignSize	public		TDataModule
Name	published		TComponent
Owner	public, read-only		TComponent
Request	public, read-only		TCustomWebDispatcher
Response	public, read-only		TCustomWebDispatcher
Tag	published	0	TComponent
VCLComObject	public		TComponent

Events

The following table lists the events of *TWebDispatcher*:

Event	Visibility	Documented in
AfterDispatch	published	TCustomWebDispatcher
BeforeDispatch	published	TCustomWebDispatcher
OnCreate	published	TDataModule
OnDestroy	published	TDataModule

Methods

The following table lists the methods of *TWebDispatcher*:

Method	Visibility	Documented in
ActionByName	public	TCustomWebDispatcher
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TCustomWebDispatcher
CreateNew	public	TDataModule
DefaultHandler	public	TObject
Destroy	public	TCustomWebDispatcher
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

TWebModule

Unit: HTTPApp

TObject • TPersistent • TComponent • TDataModule • TCustomWebDispatcher • TWebModule

TWebModule is the automatically generated Web module for a Web server application.

Description

When a new Web application is created, it automatically contains a Web module. The Web module serves as a repository for non-visual components such as *TPageProducer*, *TSession*, *TTable*, *TDataSetTableProducer*, etc. It also enables the Web server application to respond to HTTP request messages by passing the request and response objects to the appropriate action items. An application can have only one Web module.

The *TWebModule* object manages a collection of action items which know how to respond to HTTP request messages. View, add and delete the action items managed by the Web module using the Action editor which is opened by double clicking on the *TWebModule* icon.

If a data module already exists which contains many of the non-visual components and business rules for the application, the Web module can be replaced by that data module. Simply delete the Web module, add the new data module in its place, and add a *TWebDispatcher* object to the new data module.

See also

TDataSetTableProducer object, *TPageProducer* object, *TSession* object, *TTable* object, *TWebActionItems* object, *TWebDispatcher* object

Properties

The following table lists the properties of *TWebModule*:

Property	Visibility	Default	Documented in
Action	public, read-only		TCustomWebDispatcher
Actions	published		TCustomWebDispatcher
AfterDispatch	published		TCustomWebDispatcher
BeforeDispatch	published		TCustomWebDispatcher
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DesignInfo	public		TComponent
DesignOffset	public		TDataModule
DesignSize	public		TDataModule

Property	Visibility	Default	Documented in
Name	published		TComponent
Owner	public, read-only		TComponent
Request	public, read-only		TCustomWebDispatcher
Response	public, read-only		TCustomWebDispatcher
Tag	published	0	TComponent
VCLComObject	public		TComponent

Events

The following table lists the events of *TWebModule*:

Event	Visibility	Documented in
OnCreate	published	TDataModule
OnDestroy	published	TDataModule

Methods

The following table lists the methods of *TWebModule*:

Method	Visibility	Documented in
ActionByName	public	TCustomWebDispatcher
Assign	public	TPersistent
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Create	public	TWebModule
CreateNew	public	TDataModule
DefaultHandler	public	TObject
Destroy	public	TCustomWebDispatcher
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent

Method	Visibility	Documented in
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RemoveComponent	public	TComponent
SafeCallException	public	TComponent

Create

Create creates an instance of *TWebModule*.

constructor Create(AOwner: TComponent); **override**;

Description Do not call the *Create* method for *TWebModule* directly. The Web module is created automatically when a Web application project is created at design time. Every time a Web server application is run in response to an HTTP request message, a new *TWebModule* is created based on the Web module that was created at design time.

Create generates an *OnCreate* event, so that Web applications can initialize the Web module before the new HTTP request message is processed.

See also *OnCreate* event, *TCustomWebDispatcher.Create* method

TWinControl

Unit: Controls

TObject •TPersistent •TComponent •TControl • TWinControl

TWinControl is the abstract base class for all windowed controls.

Description Windowed controls are controls that

- Can receive focus while the application is running.

  Other controls may display data, but the user can use the keyboard to interact with a control only if the control is a windowed control.

W

- Can contain other controls.

  A control that contains other controls is a parent. Only a windowed control can be a parent of one or more other child controls.

- Has a window handle.

Newly created components are seldom derived directly from *TWinControl*. Base most new controls on *TCustomControl*, which provides a canvas and handles paint messages, or on any of several more specialized controls, such as *TButtonControl*, *TCustomComboBox*, *TCustomEdit*, or *TCustomListBox*.

See also *TButtonControl* object, *TCustomComboBox* object, *TCustomControl* object, *TCustomEdit* object, *TCustomListBox* object

Properties

The following table lists the properties of *TWinControl*:

Property	Visibility	Default	Documented in
Align	public	alNone	TControl
BoundsRect	public		TControl
Brush	public, read-only		TWinControl
ClientHeight	public		TControl
ClientOrigin	public, read-only		TWinControl
ClientRect	public, read-only		TWinControl
ClientWidth	public		TControl
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ControlCount	public, read-only		TWinControl
Controls	public, read-only		TWinControl
ControlState	public		TControl
ControlStyle	public		TControl
Ctl3D	protected		TWinControl
Cursor	published	crDefault	TControl
DefWndProc	protected		TWinControl
DesignInfo	public		TComponent
Enabled	public	True	TControl
Handle	public, read-only		TWinControl
Height	published		TControl
HelpContext	published	0	TWinControl
Hint	published		TControl

Property	Visibility	Default	Documented in
ImeMode	protected	imDontCare	TWinControl
ImeName	protected		TWinControl
Left	published		TControl
Name	published		TControl
Owner	public, read-only		TComponent
Parent	public		TControl
ParentCtl3D	protected	True	TWinControl
ParentWindow	public		TWinControl
ShowHint	public		TControl
Showing	public, read-only		TWinControl
TabOrder	public	–1	TWinControl
TabStop	public	False	TWinControl
Tag	published	0	TComponent
Top	published		TControl
VCLComObject	public		TComponent
Visible	public	True	TControl
Width	published		TControl
WindowHandle	protected		TWinControl
WindowProc	public		TControl

Brush

The *Brush* property determines what color and pattern is used for painting the background of the control.

property Brush: TBrush;

Description Read the *Brush* property to determine the style and color of the control's background. *Brush* is a read-only property, because a windowed control owns a single *Brush* object to handle the painting of its background. The attributes of the brush can be changed, however, by using the *Color* and *Style* properties of the *TBrush* object. Further, all of the attributes of the brush can be changed to match those of another brush, by using the *Assign* method of the *TBrush* object.

See also *TBrush* object, *TBrush.Assign* method, *TBrush.Color* property, *TBrush.Style* property

ClientOrigin

ClientOrigin is the screen coordinates (in pixels) of the top left corner of the client area of the control.

property ClientOrigin: TPoint;

Description Read *ClientOrigin* to discover where the top-left corner of the control is on the screen. *ClientOrigin* returns *X* and *Y* coordinates in a record of type *TPoint*. The *TPoint* type defines a pixel location onscreen, with the origin in the top left corner of the control. *X* specifies the horizontal coordinate of the point, *Y* specifies the vertical coordinate.

See also *ClientOrigin* property, *TPoint* type

ClientRect

ClientRect is the size (in pixels) of a control's client area.

```
property ClientRect: TRect;
```

Description Read *ClientRect* to learn the size of the client area of a control. *ClientRect* returns a rectangle defining the client area of the control. It is equivalent to *Rect*(0, 0, *ClientWidth*, *ClientHeight*).

See also *ClientHeight* property, *ClientRect* property, *ClientWidth* property, *TRect* type

ControlCount

The *ControlCount* property indicates the number of controls that are children of the windowed control.

```
property ControlCount: Integer;
```

Description Use *ControlCount* when iterating over all the children of this control. The children of the control are listed in the *Controls* property array.

ControlCount is a read-only property.

Note The value of *ControlCount* is always 1 greater than the highest *Controls* index, because the first *Controls* index is 0.

See also *Controls* property

Controls

The *Controls* property is an array of all controls that are children of the windowed control.

```
property Controls[Index: Integer]: TControl;
```

Description *Controls* is an array of all the child controls of the windowed control. All of the child controls of the windowed control list the windowed control as their *Parent* property. The *Controls* property is convenient for referring to the children of a control by number rather than name. For example, *Controls* may be used to iterate over all the children of the windowed control.

Don't confuse the *Controls* property with the *Components* property. The *Controls* property lists all the controls that are child windows of the control, while the *Components* property lists all components that are owned by the component. All components put on a form are owned by the form, and

therefore, they appear in the form's *Components* property list, even when they are child windows of a control on the form.

Controls is a read-only property. To add or delete a child control, use the *InsertControl* or *RemoveControl* methods. To move a child control from one parent to another, set the *Parent* of the child control, as that will handle both the *RemoveControl* from the original parent and the *InsertControl* to the new parent.

See also *Components* property, *ControlCount* property, *InsertControl* method, *Parent* property, *RemoveControl* method

Ctl3D

The *Ctl3D* property determines whether a control has a three-dimensional (3-D) or two-dimensional look.

```
property Ctl3D: Boolean;
```

Description Use *Ctl3D* to control the appearance of the control. If *Ctl3D* is *True*, the control has a 3-D appearance. If *Ctl3D* is *False*, the control appears flat. The default value of *Ctl3D* is *True*.

If a control's *ParentCtl3D* property is *True*, then changes in the *Ctl3D* property of the control's parent automatically changes the *Ctl3D* property of the control. When assigning a value directly to a control's *Ctl3D* property, the control's *ParentCtl3D* property is automatically set to *False*.

Note For *Ctl3D* to work with radio buttons, check boxes, and any of the common dialog boxes on Windows NT 3.51, CTL3D32.DLL must be installed in the System32 directory. Neither Windows 95 nor NT 4.0 require CTL3D32.DLL.

See also *ParentCtl3D* property

DefWndProc

The *DefWndProc* property specifies the default window procedure for the windowed control.

```
property DefWndProc: Pointer;
```

Description Use the Windows API function *CallWindowProc* with *DefWndProc* to invoke the standard Windows message handling for a window message. Use *DefWndProc* instead of the *WndProc* method to avoid any message processing introduced by the *WndProc* method.

DefWndProc fills the role for windowed controls that *DefaultHandler* fills for all objects. The *DefaultHandler* for *TWinControl* uses *DefWndProc* to pass messages to Windows for processing.

For windowed controls, *DefWndProc* is initially set to the window procedure of the window class specified in the *Params* parameter in the *CreateParams* method. Change *DefWndProc* to subclass the window class of a windowed control.

See also *CreateParams* method, *DefaultHandler* method, *WndProc* method

Handle

The *Handle* property provides access to the window handle of the control.

```
property Handle: HWND;
```

Description Use the *Handle* when making Windows API function calls that requires a window handle.

If a window handle for the control doesn't exist, the *Handle* property creates one by calling the *HandleNeeded* method when your application reads the *Handle* property value. Therefore, do not use the *Handle* property during component creation or streaming.

Handle is a read-only property.

See also *HandleNeeded* method

HelpContext

The *HelpContext* provides a context number for use in calling context-sensitive online Help.

```
property HelpContext: THelpContext;
```

Description Assign a value to *HelpContext* to determine which help screen appears when the user presses *F1* to request help.

Each screen in the Help system should have a unique context number. When a control is selected in the application, pressing *F1* displays a Help screen. Which Help screen appears depends on the value of the *HelpContext* property.

If *HelpContext* is zero, then the control inherits the help context of its parent control. For example, if a button with a *HelpContext* of zero is contained in a form with a non-zero help context and *F1* is pressed, the context screen for the form will be shown.

See also *HelpContext* method, *HelpFile* property, *HelpJump* method, *OnHelp* event

ImeMode

ImeMode determines the behavior of the input method editor (IME).

```
type TImeMode = (imDisable, imClose, imOpen, imDontCare, imSAlpha, imAlpha, imHira,
    imSKata, imKata, imChinese, imSHanguel, imHanguel);

property ImeMode: TImeMode;
```

Description Set *ImeMode* to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks

all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the Delphi application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an *ImeMode* of *imClose*, as no conversion is necessary for numeric input.

ImeMode can have one of the following values:

Value	Meaning
imDisable	Shut down the IME. *imDisable* has no effect on Chinese, Taiwanese, or Korean IMEs.
imClose	Close the IME conversion window, but leave the IME running in the background. The IME can be re-activated by a hotkey combination.
imOpen	Open the IME conversion window. The conversion mode is the last conversion mode used by the IME.
imDontCare	Launch the IME if it is disabled. The conversion mode is the last conversion mode used by the IME.
imSAlpha	Open the IME conversion window and set the conversion mode to accept single-width Roman alphabet input.
imAlpha	Open the IME conversion window and set the conversion mode to accept double-width Roman alphabet input.
imHira	Open the IME conversion window and set the conversion mode to double-width Hiragana. *imHira* is only available for Japanese IMEs.
imSKata	Open the IME conversion window and set the conversion mode to single-width Katakana (Hankaku Katakana). *imSKata* is only available for Japanese IMEs.
imKata	Open the IME conversion window and set the conversion mode to double-width Katakana (Zenkaku Katakana). *imKata* is only available for Japanese IMEs.
imChinese	Open the IME conversion window and set the conversion mode to double-width Chinese. *imChinese* is only available for Chinese IMEs.
imSHanguel	Open the IME conversion window and set the conversion mode to single-width Hanguel. *imSHanguel* is only available for Korean IMEs.
imHanguel	Open the IME conversion window and set the conversion mode to double-width Hanguel. *imHanguel* is only available for Korean IMEs.

Note The value of *ImeMode* only takes effect when the control receives focus. To change the value of *ImeMode* after the control has gotten input focus, call the *SetIme* method.

See also *ImeName* property, *ResetIme* method, *ResetImeComposition* method, *SetIme* method, *SetImeCompositionWindow* method

W

ImeName

ImeName specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

```
TImeName = type string;

property ImeName: TImeName;
```

Description Set *ImeName* to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the Delphi application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global *Screen* variable.

If *ImeName* specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note The value of *ImeName* only takes effect when the control receives focus. To change the value of *ImeName* after the control has gotten input focus, call the *SetIme* method.

See also *ImeMode* property, *ResetIme* method, *SetIme* method, *TScreen.DefaultIme* property, *TScreen.Imes* property

ParentCtl3D

The *ParentCtl3D* property determines where a component looks to determine if it should appear three dimensional.

```
property ParentCtl3D: Boolean;
```

Description Use *ParentCtl3D* to ensure that all the components on a form have a uniform appearance. For example, to make all components on a form appear three dimensional, set the form's *Ctl3D* property to *True* and each control's *ParentCtl3D* property to *True*. Not only will all components have a three-dimensional appearance, but to change to a uniform two-dimensional appearance, you only have to change the *Ctl3D* property of the form and all the components will become two dimensional.

If *ParentCtl3D* is *True*, the component uses the value of its parent component's *Ctl3D* property. If *ParentCtl3D* is *False*, the control uses the value of its own *Ctl3D* property. The default value is *True*.

To specify either a 2D or 3D appearance for a particular component, set *Ctl3D* to *True* (for 3D) or *False* (for 2D). When setting the Ctl3D property of a control, *ParentCtl3D* automatically gets set to *False*.

See also *Ctl3D* property

ParentWindow

ParentWindow is the window handle of a non-VCL parent window that contains the control.

```
property ParentWindow: HWnd;
```

Description Set *ParentWindow* to host a windowed control in an non-VCL parent window. Setting *ParentWindow* moves the control so that it is contained in the specified window. For example, *TActiveXControl* objects use *ParentWindow* to insert a control in an ActiveX container window. *ParentWindow* is set automatically when a control is created using the *CreateParented* method.

Use *ParentWindow* with windowed controls that are packaged in a DLL so that they may be used by a VB, OWL or MFC application.

A *TWinControl* object can have either a *Parent* property or a *ParentWindow* property, but not both. If the *Parent* property of a control is not **nil**, the value of *ParentWindow* is ignored.

See also *CreateParented* method, *Parent* property, *TActiveXControl* object

Showing

The *Showing* property specifies whether the control is currently showing on the screen.

```
property Showing: Boolean;
```

Description Use the *Showing* property to determine if the user can see the control on the screen, assuming that another control is not obstructing it from view.

If the *Visible* properties of a component and all the parents in its parent hierarchy are *True*, *Showing* is *True*. If one of the parents containing the component has a *Visible* property value of *False*, *Showing* is *False*.

Showing is a read-only property.

See also *Visible* property

TabOrder

The *TabOrder* property indicates the position of the control in its parent's tab order.

```
type TTabOrder = -1..32767;
property TabOrder: TTabOrder;
```

Description *TabOrder* is the order in which child windows are visited when the user presses the *Tab* key. The control with the *TabOrder* value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a *TabOrder* value of 0, the second is 1, the third is 2, and so on. Change this by changing the *TabOrder* property. A control with a *TabOrder* of –1 is outside the tab order, and cannot be reached by pressing the *Tab* key.

Each control has a unique tab-order value within its parent. If you change the *TabOrder* property value of one control to be the same as the value of a second control, the *TabOrder* value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's *TabOrder* property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning *TabOrder* a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of *TabOrder*, but instead is given the number that assures the control is the last in the tab order.

Note *TabOrder* is meaningful only if the *TabStop* property is *True* and if the control has a parent. The *TabOrder* property of a form is not used unless the form assigns another form to be its parent.

See also *FindNextControl* method, *TabStop* property

TabStop

The *TabStop* property determines if the user can tab to a control.

```
property TabStop: Boolean;
```

Description Use the *TabStop* to allow or disallow access to the control using the *Tab* key.

If *TabStop* is *True*, the control is in the tab order. If *TabStop* is *False*, the control is not in the tab order and the user can't press the *Tab* key to move to the control.

Note *TabStop* is not meaningful for a form unless the form assigns another form to be its parent.

See also *TabOrder* property

WindowHandle

The *WindowHandle* provides access to a window handle for the control.

```
property WindowHandle: Hwnd;
```

Description The *WindowHandle* property provides access to the same window handle as the *Handle* property, but *WindowHandle* is protected, and therefore only accessible to code inside the control.

The advantage to using *WindowHandle* is that it can be read and written to, while the *Handle* property is read-only. *WindowHandle* is intended to be used when implementing methods that need to change the value of the controls window handle. Unlike the *Handle* property, reading the value of *WindowHandle* doesn't automatically create a valid handle. Reading *WindowHandle* can return a zero value.

See also *Handle* property, *HandleNeeded* method

Events

The following table lists the events of *TWinControl*:

Event	Visibility	Documented in
OnEnter	protected	TWinControl
OnExit	protected	TWinControl
OnKeyDown	protected	TWinControl
OnKeyPress	protected	TWinControl
OnKeyUp	protected	TWinControl

OnEnter

The *OnEnter* event occurs when a control receives the input focus.

```
type TNotifyEvent = procedure(Sender: TObject) of object;
```

```
property OnEnter: TNotifyEvent;
```

Description Use the *OnEnter* event handler to cause any special processing to occur when a windowed component becomes active.

The *OnEnter* event does not occur when switching between forms or between another Windows application and the application that includes the windowed control.

When switching between controls in separate container controls such as the *TPanel* and the *TGroupBox* controls, an *OnEnter* event occurs for the container before the *OnEnter* event of the contained control.

Similarly, an *OnExit* event of the container occurs after the *OnExit* event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an *OnExit* event of the button occurs, followed by an *OnEnter* event on the group box, and finally an *OnEnter* event on the radio button that was clicked. If the user then clicks on the OK button, an *OnExit* event for the radio button occurs followed by an *OnExit* event for the group box, and then the button's *OnEnter* event occurs.

See also *OnExit* event, *TForm.ActiveControl* property, *TForm.OnActivate* event

OnExit

The *OnExit* event occurs when the input focus shifts away from one control to another.

```
type TNotifyEvent = procedure(Sender: TObject) of object;
```

```
property OnExit: TNotifyEvent;
```

Description Use the *OnExit* event handler to provide special processing when the control ceases to be active.

The *OnExit* event does not occur when switching between forms or between another Windows application and your application.

When switching between controls in separate container controls such as the *TPanel* and the *TGroupBox* controls, an *OnExit* event occurs for the control inside the container before the *OnExit* event of the container.

Similarly, an *OnEnter* event of the container occurs before the *OnEnter* event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an *OnExit* event of the button occurs, followed by an *OnEnter* event on the group box, and finally an *OnEnter* event on the radio button that was clicked. If the user then clicks on the OK button, an *OnExit* event for the radio button occurs followed by an *OnExit* event for the group box, and then the button's *OnEnter* event occurs.

Note The value of the *ActiveControl* property updates before the *OnExit* event occurs.

See also *OnEnter* event, *TForm.ActiveControl* property

OnKeyDown

The *OnKeyDown* event occurs when a user presses any key while the control has focus.

```
type
  TShiftState = set of (ssShift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle, ssDouble);
  TKeyEvent = procedure (Sender: TObject; var Key: Word; Shift: TShiftState) of object;

property OnKeyDown: TKeyEvent;
```

Description Use the *OnKeyDown* event handler to specify special processing to occur when a key is pressed. The *OnKeyDown* handler can respond to all keyboard keys, including function keys and keys combined with the *Shift, Alt,* and *Ctrl* keys, and pressed mouse buttons.

The *TKeyEvent* type points to a method that handles keyboard events. The *Key* parameter is the key on the keyboard. For non-alphanumeric keys, use WinAPI virtual key codes to determine the key pressed. For more information, search for *VkKeyScan* and *VkKeyScanEx* in the Win32 Developer's Reference (WIN32.HLP).

The *Shift* parameter includes zero or more of the following values:

State	Meaning
ssShift	The Shift key is held down.
ssAlt	The Alt key is held down.
ssCtrl	The Ctrl key is held down.
ssLeft	The left mouse button is held down.
ssMiddle	The middle mouse button is held down.
ssDouble	The mouse was double-clicked (not used for keystrokes).

See also *KeyPreview* property, *OnKeyPress* event, *OnKeyUp* event

OnKeyPress

The *OnKeyPress* event occurs when a user presses a single character key.

```
type TKeyPressEvent = procedure (Sender: TObject; var Key: Char) of object;

property OnKeyPress: TKeyPressEvent;
```

Description Use the *OnKeyPress* event handler to make something happen as a result of a single character key press.

The *Key* parameter in the *OnKeyPress* event handler is of type *Char*; therefore, the *OnKeyPress* event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII *Char* value (*Shift* or *F1*, for example) don't generate an *OnKeyPress* event. Key combinations (such as *Shift+A*), generate only one *OnKeyPress* event (for this example, *Shift+A* results in a *Key* value of "A" if *Caps Lock* is off). To respond to non-ASCII keys or key combinations, use the *OnKeyDown* or *OnKeyUp* event handlers.

See also *OnKeyDown* event, *OnKeyUp* event, *TForm.KeyPreview* property

OnKeyUp

The *OnKeyUp* event occurs when the user releases a key that has been pressed.

```
type
  TShiftState = set of (ssShift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle, ssDouble);
  TKeyEvent = procedure (Sender: TObject; var Key: Word; Shift: TShiftState) of object;

property OnKeyUp: TKeyEvent;
```

Description Use the *OnKeyUp* event handler to provide special processing that occurs when a key is released. The *OnKeyUp* handler can respond to all keyboard keys, including function keys and keys combined with the *Shift, Alt,* and *Ctrl* keys.

The *TKeyEvent* type points to a method that handles keyboard events. The *Key* parameter is the key on the keyboard. For non-alphanumeric keys, you

must use WinAPI virtual key codes to determine the key pressed. For more information, search for *VkKeyScan* and *VkKeyScanEx* in the Win32 Developer's Reference (WIN32.HLP).

The *Shift* parameter includes zero or more of the following values:

State	Meaning
ssShift	The Shift key is held down.
ssAlt	The Alt key is held down.
ssCtrl	The Ctrl key is held down.
ssLeft	The left mouse button is held down.
ssMiddle	The middle mouse button is held down.
ssDouble	The mouse was double-clicked (not used for keystrokes).

See also *OnKeyDown* event, *OnKeyPress* event

Methods

The following table lists the methods of *TWinControl*:

Method	Visibility	Documented in
AlignControls	protected	TWinControl
Assign	public	TPersistent
BeginDrag	public	TControl
BringToFront	public	TControl
Broadcast	public	TWinControl
CanFocus	public	TWinControl
ChangeScale	protected	TWinControl
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
ClientToScreen	public	TControl
ContainsControl	public	TWinControl
ControlAtPos	public	TWinControl
Create	public	TWinControl
CreateHandle	protected	TWinControl
CreateParams	protected	TWinControl
CreateParented	public	TWinControl
CreateSubClass	protected	TWinControl
CreateWindowHandle	protected	TWinControl
CreateWnd	protected	TWinControl

Method	Visibility	Documented in
DefaultHandler	protected	TWinControl
Destroy	public	TWinControl
DestroyComponents	public	TComponent
DestroyHandle	protected	TWinControl
Destroying	public	TComponent
DestroyWindowHandle	protected	TWinControl
DestroyWnd	protected	TWinControl
DisableAlign	public	TWinControl
Dispatch	public	TObject
DoEnter	protected	TWinControl
DoExit	protected	TWinControl
DoKeyDown	protected	TWinControl
DoKeyPress	protected	TWinControl
DoKeyUp	protected	TWinControl
DragDrop	public	TControl
Dragging	public	TControl
EnableAlign	public	TWinControl
EndDrag	public	TControl
FieldAddress	public	TObject
FindComponent	public	TComponent
FindNextControl	protected	TWinControl
FixupTabList	protected	TWinControl
Focused	public	TWinControl
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetChildren	protected	TWinControl
GetClientOrigin	protected	TWinControl
GetClientRect	protected	TWinControl
GetDeviceContext	protected	TWinControl
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent
GetParentHandle	protected	TWinControl
GetTabOrderList	public	TWinControl
GetTextBuf	public	TControl
GetTextLen	public	TControl
GetTopParentHandle	protected	TWinControl
HandleAllocated	public	TWinControl

Method	Visibility	Documented in
HandleNeeded	public	TWinControl
HasParent	public	TComponent
Hide	public	TControl
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InsertControl	public	TWinControl
InstanceSize	public	TObject
Invalidate	public	TWinControl
IsControlMouseMsg	protected	TWinControl
KeyDown	protected	TWinControl
KeyPress	protected	TWinControl
KeyUp	protected	TWinControl
MainWndProc	protected	TWinControl
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
NotifyControls	protected	TWinControl
PaintControls	protected	TWinControl
PaintHandler	protected	TWinControl
PaintTo	public	TWinControl
PaintWindow	protected	TWinControl
PaletteChanged	protected	TWinControl
Perform	public	TControl
ReadState	protected	TWinControl
Realign	public	TWinControl
RecreateWnd	protected	TWinControl
Refresh	public	TControl
RemoveComponent	public	TComponent
RemoveControl	public	TWinControl
Repaint	public	TWinControl
ResetIme	protected	TWinControl
ResetImeComposition	protected	TWinControl
SafeCallException	public	TComponent
ScaleBy	public	TWinControl
ScaleControls	protected	TWinControl
ScreenToClient	public	TControl
ScrollBy	public	TWinControl
SelectFirst	protected	TWinControl
SelectNext	protected	TWinControl
SendToBack	public	TControl
SetBounds	public	TWinControl

Method	Visibility	Documented in
SetChildOrder	protected	TWinControl
SetFocus	public	TWinControl
SetIme	protected	TWinControl
SetImeCompositionWindow	protected	TWinControl
SetTextBuf	public	TControl
SetZOrder	protected	TWinControl
Show	public	TControl
ShowControl	protected	TWinControl
Update	public	TWinControl
UpdateControlState	public	TWinControl
WndProc	protected	TWinControl

AlignControls

AlignControls aligns any controls for which the control is the parent within a specified area of the control.

```
procedure AlignControls(AControl: TControl; var Rect: TRect); virtual;
```

Description Use *AlignControls* to align all controls within an area of a form, panel, group box, scroll box, or any windowed control that contains several other controls. Specify the area in which to align the controls as the value of the *Rect* parameter.

AlignControls uses the *Align* property value for each child control to determine how to align it. The *AControl* parameter can be **nil**. If you specify a control in *AControl*, that control takes precedence in alignment over other, similarly-aligned controls.

See also *Align* property, *TRect* type

Broadcast

Broadcast sends a message to each of the child controls of the windowed control.

```
procedure Broadcast(var Message);
```

Description Use *Broadcast* when you want to send the same message to each of the child controls contained within the windowed control. Specify the message to be passed as the value of the *Message* parameter.

See also *NotifyControls* method

W

CanFocus

CanFocus determines whether a control can receive focus.

```
function CanFocus: Boolean;
```

Description Call *CanFocus* to find out if the control can receive input from the user.

CanFocus returns *True* if both the control and its parent(s) have their *Visible* and *Enabled* properties set to *True*. If any of the *Visible* and *Enabled* properties of the control or the controls in which it is contained are not *True*, then *CanFocus* returns *False*. Override the *CanFocus* method to provide further restrictions to when the control can receive focus in descendant objects.

See also *Enabled* property, *Parent* property, *Visible* property

ChangeScale

ChangeScale resizes and repositions the control, rescaling the children of the control before the control itself is rescaled.

```
procedure ChangeScale(M, D: Integer); override;
```

Description *ChangeScale* can be used to change the scale of a form and all its controls for a different screen resolution or font size. Because *ChangeScale* modifies the control's *Top*, *Left*, *Width*, and *Height* properties, it changes the position of the control and its children as well as their size.

The *M* parameter is the multiplier and the *D* parameter is the divisor. For example, to change a control and its children to be 75% of their original sizes, specify the value of *M* as 75, and the value of *D* as 100 (75/100). Alternately, specify the value of *M* as 3, and the value of *D* as 4 (3/4). Both fractions are equal and result in the controls being scaled by the same amount, 75%.

To make the control and its children 33% larger than their previous sizes, specify the value of *M* as 133 and the value of *D* as 100 (133/100), or specify the value of M as 4, and the value of D as 3 (4/3).

Note Because control sizes and coordinates are integers, scaling them by the *M/D* ratio incurs some degree of roundoff error. Calling *ChangeScale* repeatedly can mean that at some point the control disappears (when the roundoff errors make the control smaller), or at some point it creeps off the form (when the roundoff errors move the location of control). If rescaling a form repeatedly, destroy and reload the form after several rescalings to eliminate the accumulating roundoff error.

See also *ScaleBy* method

ContainsControl

ContainsControl indicates whether a specified control exists within the control.

```
function ContainsControl(Control: TControl): Boolean;
```

Description Use *ContainsControl* to find out if the a particular control exists within this control. *ContainsControl* returns *True* if the specified value of the *Control* parameter is a child control of the windowed control. The *Control* parameter

need not be an immediate child (in the *Controls* property), but may be contained in a child of the windowed control, or in a child of a child of the windowed control, for indefinitely many levels deep.

If the method returns *False*, the specified control is not within the control.

See also *ContainsAtPos* method, *Controls* property, *Parent* property

ControlAtPos

ControlAtPos returns the child control located at a specified position within the control.

```
function ControlAtPos(const Pos: TPoint; AllowDisabled: Boolean): TControl;
```

Description Use *ControlAtPos* when you want to know which child control is at the specified location within the control. *ControlAtPos* will return an immediate child of the windowed control; that is, one of the entries of the *Controls* property, that has the windowed control for its *Parent* property.

Specify the position in client coordinates as the value of the *Pos* parameter. *Pos* can be anywhere within the boundaries of the child control, not just the upper left corner.

If there is no control at the specified position, *ControlAtPos* returns **nil**. The *AllowDisabled* parameter determines whether the search for controls includes disabled controls.

See also *ContainsControl* method, *Controls* property, *Parent* property, *ScreenToClient* method

Create

Create creates an instance of *TWinControl*.

```
constructor Create(AOwner: TComponent); override;
```

Description Call *Create* to construct and initialize a new windowed control and insert the newly-constructed control into its owner, as specified by the *AOwner* parameter. *Create* inserts the control in the owner by calling the owner's *InsertComponent* method.

After calling the inherited constructor, *Create*

- Creates and initializes the *Brush*.
- Initializes the *ParentCtl3D* property to *True*.
- Sets the *TabOrder* property to –1.

Most controls override *Create* to initialize their unique properties. Objects that override the *Create method* must always call the inherited *Create* method first, and then proceed with the component-specific initialization. Specify the **override** directive when overriding the *Create* method.

If a component's *Create* method allocates resources or memory, override the *Destroy* method to free those resources.

See also *Brush* property, *TabOrder* property, *TComponent.InsertComponent* method, *TControl.Create* method

CreateHandle

CreateHandle generates a handle for the *Handle* property of the control if it does not already have one.

```
procedure CreateHandle; virtual;
```

Description *CreateHandle* calls the *CreateWnd* method to create the window handle if the control has a parent. *CreateHandle* creates a handle in the parent control before it creates a handle for the control itself.

Note If the *Parent* property is **nil**, *CreateHandle* raises an exception.

See also *CreateWindowHandle* method, *CreateWnd* method, *Handle* property, *Parent* property

CreateParams

CreateParams initializes a window-creation parameter record passed in the *Params* parameter.

```
type
  TCreateParams = record
    Caption: PChar;
    Style: Longint;
    ExStyle: Longint;
    X, Y: Integer;
    Width, Height: Integer;
    WndParent: HWND;
    Param: Pointer
    WindowClass: TWndClass;
    WinClassName: array[0..63] of Char;
  end;

procedure CreateParams(var Params: TCreateParams); virtual;
```

Description The *CreateWnd* method calls *CreateParams* to initialize the parameters it passes to *CreateWindowHandle*. Override *CreateParams to* customize the way a control creates its Windows representation. When overriding *CreateParams*, always call the inherited method first to set the default values, then make any desired adjustments.

Specify the parameter record as the value of the *Params* parameter. The *TCreateParams* type is a data structure holding information needed when telling Windows to create a window handle. The fields of a *TCreateParams* record become the parameters to a call to the *CreateWindowEx* API function.

TWinControl implements *CreateParams* by setting all the fields of *Params* to generic base values. A number of the standard controls override *CreateParams* to change one or more of the default values in *Params*.

See also *CreateWindowHandle* method, *CreateWnd* method

CreateParented

CreateParented creates and initializes a control as the child of a specified non-VCL window.

```
constructor CreateParented(ParentWindow: HWnd);
```

Description Call *CreateParented* to embed a windowed control in a non-VCL parent window. *TActiveXControl* objects call *CreateParented* to create an ActiveX control as a child of the host application's client site window.

CreateParented calls the *Create* method to initialize the control normally, but with the *ParentWindow* property set to the *ParentWindow* parameter, and the *Owner* property set to **nil**.

See also *Create* method, *Owner* property, *ParentWindow* property

CreateSubClass

CreateSubClass creates a Delphi windowed control derived from an existing Windows window class.

```
type
  TCreateParams = record
    Caption: PChar;
    Style: Longint;
    ExStyle: Longint;
    X, Y: Integer;
    Width, Height: Integer;
    WndParent: HWND;
    Param: Pointer
    WindowClass: TWndClass;
    WinClassName: array[0..63] of Char;
  end;

procedure CreateSubClass(var Params: TCreateParams; ControlClassName: PChar);
```

Description Use *CreateSubClass* when you want to create a *TCreateParams* record to pass to a call to the *CreateWindowEx* API function. The *TCreateParams* type is a data structure holding information needed when telling Windows to create a window handle.

Call *CreateSubClass* in the *CreateParams* method of a subclassed control, after calling the inherited *CreateParams*. Specify the parameter record as the value of the *Params* parameter. *CreateSubClass* allows VCL controls to create standard Windows controls.

See also *CreateParams* method, *CreateWindowHandle* method

CreateWindowHandle

CreateWindowHandle creates a window handle for the control.

```
type
  TCreateParams = record
    Caption: PChar;
    Style: Longint;
    ExStyle: Longint;
    X, Y: Integer;
    Width, Height: Integer;
    WndParent: HWND;
    Param: Pointer
    WindowClass: TWndClass;
    WinClassName: array[0..63] of Char;
  end;

procedure CreateWindowHandle(const Params: TCreateParams); virtual;
```

Description Call *CreateWindowHandle* to create a window handle for the control. *CreateWindowHandle* creates the window handle by calling the *CreateWindowEx* API function, passing parameters from the record passed in the *Params* parameter.

The *TCreateParams* type is a data structure holding information needed when telling Windows to create a window handle.

See also *CreateParams* method, *CreateWnd* method

CreateWnd

CreateWnd creates the window used by the windowed-control component.

```
procedure CreateWnd; virtual;
```

Description *CreateWnd* first calls the *CreateParams* method to initialize the window-creation parameters, then calls *CreateWindowHandle* to create the window handle for the control. *CreateWnd* adjusts the size of the newly-created window and then sets the control's font by calling the *Perform* method, passing the WM_SETFONT message.

See also *CreateParams* method, *CreateWindowHandle* method, *Perform* method

DefaultHandler

DefaultHandler provides message handling for all messages that the control does not fully process itself.

```
procedure DefaultHandler(var Message); virtual;
```

Description Override *DefaultHandler* to change the default message handling for the control. The *Message* parameter can be cast to a *TMessage* type, to obtain the *WParam*, *LParam*, and *Result* of the message. If the *Result* of the message is non-zero, the message has already been handled. Set the *Result* field to a non-zero value to prevent further processing of the message by the inherited method.

Calling **inherited** in a message-handling method results in a call to the ancestor's *DefaultHandler* method if that ancestor does not specify a handler for the message being handled.

TWinControl overrides *TControl.DefaultHandler* to handle messages for all its descendant types. *DefaultHandler* passes any otherwise-unhandled messages to the control's window procedure by calling the *CallWindowProc* API function.

See also *TControl.DefaultHandler*

Destroy

Destroy destroys an instance of *TWinControl*.

```
destructor Destroy; override;
```

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the *TWinControl* object is not already freed, and only then calls *Destroy*.

See also *Free* method, *TControl.Destroy* method

DestroyHandle

DestroyHandle destroys the windowed control's window handle without destroying the control.

```
procedure DestroyHandle;
```

Description Call *DestroyHandle* to dispose of the handle, but leave the control intact. The control can later recreate the handle if needed. *DestroyHandle* is the converse operation to *CreateHandle*. Applications should call the high-level *CreateHandle* and *DestroyHandle* methods, rather than the lower-level methods of *CreateWnd* and *DestroyWnd*, whenever possible.

If the control has windowed controls as child controls, *DestroyHandle* calls each of their *DestroyHandle* methods before calling *DestroyWnd* to destroy its own handle.

See also *CreateHandle* method, *DestroyWnd* method

W

DestroyWindowHandle

DestroyWindowHandle calls the Windows API function *DestroyWindow* to destroy the window handle created in the *CreateWindowHandle* method.

procedure DestroyWindowHandle; **virtual**;

Description Call *DestroyWindowHandle* to dispose of the window handle for the control.

TWinControl's *Destroy* method calls *DestroyWindowHandle* to destroy any window handle associated with a windowed control before destroying the object. The *DestroyWnd* method also calls *DestroyWindowHandle*.

See also *CreateWindowHandle* method, *Destroy* method, *DestroyWnd* method

DestroyWnd

DestroyWnd destroys the windowed control's window handle.

procedure DestroyWnd; **virtual**;

Description Before the window handle is destroyed, *DestroyWnd* saves a copy of the control's text in internal storage, frees any device contexts, and finally calls *DestroyWindowHandle*.

See also *DestroyWindowHandle* method

DisableAlign

DisableAlign temporarily disables the realigning of controls within the windowed control.

procedure DisableAlign;

Description Call *DisableAlign* to prevent child controls from realigning. You might want to disable aligning while performing multiple manipulations of controls, such as reading from a form file or scaling. Each call to *DisableAlign* must be followed by a corresponding call to *EnableAlign*.

DisableAlign increments a reference count that *EnableAlign* later decrements. When the reference count reaches zero, *EnableAlign* performs any needed realignments.

See also *EnableAlign* method

DoEnter

DoEnter is the protected implementation method for a windowed control's *OnEnter* event.

procedure DoEnter; **dynamic**;

Description The *DoEnter* method does nothing except call any event handler attached to the *OnEnter* event. Override *DoEnter* to provide other responses in addition to the inherited event-handler call.

See also *DoExit* method, *OnEnter* event

DoExit

DoExit is the protected implementation method for a windowed control's *OnExit* event.

```
procedure DoExit; dynamic;
```

Description The *DoExit* method does nothing except call any event handler attached to the *OnExit* event. Override *DoExit* to provide other responses in addition to the inherited event-handler call.

See also *DoEnter* method, *OnExit* event

DoKeyDown

DoKeyDown is a protected method that performs some preprocessing before calling the *KeyDown* method that implements the *OnKeyDown* event.

```
type
  TWMKey = record
    Msg: Cardinal;
    CharCode: Word;
    KeyData: Longint;
    Result: Longint;
  end;

function DoKeyDown(var Message: TWMKey): Boolean;
```

Description Use *DoKeyDown* to determine if the application should continue processing key-down messages passed in the *Message* parameter. A return value of *True* indicates that the key-down occurrence has been handled completely, and no further processing is needed. A return value of *False* indicates that the application should continue passing the key-down message through the inherited processing.

DoKeyDown checks whether the *KeyPreview* property of the parent form is *True*; and if so, allows the form to preemptively process key-down messages before they are handled by the windowed control. If the form does not handle the message, *DoKeyDown* translates the message parameters into the appropriate types and calls *KeyDown*, which in turn calls the *OnKeyDown* event handler, if any.

The *TWMKey* type is the message record for all Windows keyboard messages.

See also *DoKeyPress* method, *DoKeyUp* method, *KeyDown* method, *OnKeyDown* event, *TForm.KeyPreview* property

W

DoKeyPress

DoKeyPress is a protected method that performs some preprocessing before calling the *KeyPress* method that implements the *OnKeyPress* event.

```
type
  TWMKey = record
    Msg: Cardinal;
    CharCode: Word;
    KeyData: Longint;
    Result: Longint;
  end;

function DoKeyPress(var Message: TWMKey): Boolean;
```

Description Use the return value from *DoKeyPress* to determine whether the application should continue processing the key-press message passed in the *Message* parameter. A return value of *True* indicates that the key-press occurrence has been handled completely, and no further processing is needed. A return value of *False* indicates that the application should continue passing the key-press message through the inherited processing.

DoKeyPress checks whether the *KeyPreview* property of the parent form is *True*, and if so, allows the form to preemptively process key-press messages before they are handled by the windowed control.If the form does not handle the message, *DoKeyPress* translates the message parameters into the appropriate types and calls *KeyPress*, which in turn calls the *OnKeyPress* event handler, if any.

The *TWMKey* type is the message record for all Windows keyboard messages.

See also *DoKeyDown* method, *DoKeyUp* method, *KeyPress* method, *OnKeyPress* event, *TForm.KeyPreview* property

DoKeyUp

DoKeyUp is a protected method that performs some preprocessing before calling the *KeyUp* method that implements the *OnKeyUp* event.

```
type
  TWMKey = record
    Msg: Cardinal;
    CharCode: Word;
    KeyData: Longint;
    Result: Longint;
  end;

function DoKeyUp(var Message: TWMKey): Boolean;
```

Description Use the return value from *DoKeyUp* to determine whether the application should continue processing the key-up message passed in the *Message* parameter. A return value of *True* indicates that the key-up occurrence has been handled completely, and no further processing is needed. A return value of *False* indicates that the application should continue passing the key-up message through the inherited processing.

DoKeyUp checks whether the *KeyPreview* property of the parent form is *True*, and if so, allows the form to preemptively process key-up messages before they are handled by the windowed control. If the form does not handle the message, *DoKeyUp* translates the message parameters into the appropriate types and calls *KeyUp*, which in turn calls the *OnKeyUp* event handler, if any.

The *TWMKey* type is the message record for all Windows keyboard messages.

See also *DoKeyDown* method, *KeyUp* method, *OnKeyUp* event, *TForm.KeyPreview* property

EnableAlign

EnableAlign decrements the reference count incremented by a call to the *DisableAlign* method, eventually realigning the child controls of the windowed control.

```
procedure EnableAlign;
```

Description Call *EnableAlign* to allow child controls within the control to realign again after they were prevented from realigning by a call to *DisableAlign*.

Each time the *DisableAlign* method is called, it increments a reference count. Each time *EnableAlign* is called, it decrements the same reference count. When the reference count reaches zero, *EnableAlign* calls the *Realign* method to perform any pending realignments.

Be sure to pair each call to *DisableAlign* with a call to *EnableAlign*. If an exception could be raised after the call to *DisableAlign*, use a **try** .. **finally** statement to ensure that the corresponding call to *EnableAlign* is executed.

See also *DisableAlign* method

FindNextControl

FindNextControl returns the windowed control's next child control in the tab order after the specified control.

```
function FindNextControl(CurControl: TWinControl;
  GoForward, CheckTabStop, CheckParent: Boolean): TWinControl;
```

Description Call *FindNextControl* to find the next child control in the tab order after *CurControl*. If *CurControl* is not a child of the windowed control, *FindNextControl* returns the first child control in the tab order.

The *GoForward* parameter controls the direction of the search. If *GoForward* is *True*, *FindNextControl* searches forward through the child controls in tab order. If Go *Forward* is *False*, *FindNextControl* searches backward through the controls.

The *CheckTabStop* and *CheckParent* parameters control whether *FindNextControl* performs certain checks on the controls it finds. If *CheckTabStop* is *True*, the returned control must have its *TabStop* property set

W

to *True*. If *CheckParent* is *True*, the returned control's *Parent* property must indicate the windowed control.

FindNextControl calls the *GetTabOrderList* method to build its list of possible "next" controls.

See also *GetTabOrderList* method, *TabOrder* property, *TabStop* property

FixupTabList

FixupTabList sorts the child controls of this control by their tab order.

```
procedure FixupTabList;
```

Description Applications should not call *FixupTabList directly*. Delphi uses this method in its implementation of the *ReadState* method to initialize the tab order list based on the *TabOrder* properties of the child controls being read from the stream.

See also *ReadState* property, *TabOrder* property

Focused

Focused determines whether a windowed control has input focus.

```
function Focused: Boolean;
```

Description Use the *Focused* method to see if the control is the active control. When *Focused* is *True*, the control has the focus. If *Focused* is *False*, the user cannot interact with the control.

See also *TForm.ActiveControl* property, *TScreen.ActiveControl* property

GetChildren

GetChildren calls the specified method for each child of the windowed control.

```
type TGetChildProc = procedure (Child: TComponent) of object;
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

Description *GetChildren* is called by the VCL streaming system. Applications seldom need to call this routine. Some specialized controls may override *GetChildren* to limit which child controls are written to the stream. When overriding *GetChildren*, call the procedure passed as the *Proc* parameter for every child control that should be written to the stream.

See also *Parent* property

GetClientOrigin

GetClientOrigin is a protected access method for the *ClientOrigin* property.

```
function GetClientOrigin: TPoint; override;
```

Description Call *GetClientOrigin* to find the origin of the client area of the control.

GetClientOrigin returns a point indicating the position of the top-left corner of the control in screen coordinates. *GetClientOrigin* calls the Windows API function *ClientToScreen* to return the position of top-left corner of the windowed control on the screen in screen coordinates.

See also *ClientOrigin* property, *ClientToScreen* method, *GetClientRect* method, *TPoint* type

GetClientRect

GetClientRect is a protected access method for the *ClientRect* property.

```
function GetClientRect: TRect; override;
```

Description Call *GetClientRect* to return the rectangle that defines the client area of the control. *GetClientRect* returns a rectangle with the rectangle's *Top* and *Left* fields set to zero, and its *Bottom* and *Right* fields set to the control's *ClientHeight* and *ClientWidth*, respectively. *GetClientRect* calls the Windows API function *GetClientRect*.

See also *ClientHeight* property, *ClientWidth* property, *GetClientOrigin* method, *TRect* type

GetDeviceContext

GetDeviceContext returns a device context for the control.

```
function GetDeviceContext(var WindowHandle: HWnd): HDC; override;
```

Description Call *GetDeviceContext* to return a device context for the control.

GetDeviceContext calls the Windows API function *GetDC*, passing the windowed control's *Handle* property. If the call is unsuccessful, the *EOutOfResources* exception is raised.

See also *EOutOfResources* object, *Handle* property

GetParentHandle

GetParentHandle returns the window handle for the window that contains the control.

```
function GetParentHandle: HWnd;
```

Description Use the handle returned by *GetParentHandle* as a parameter to Windows API function calls that need the window handle for the parent window of the control. If the *Parent* property is not **nil**, *GetParentHandle* returns the *Handle* property of the parent control. If the *Parent* property is **nil**, *GetParentHandle* returns the value of *ParentWindow*.

See also *GetTopParentHandle* method, *Handle* property, *Parent* property, *ParentWindow* property

GetTabOrderList

GetTabOrderList builds a list of controls in tab order.

```
procedure GetTabOrderList(List: TList);
```

Description Call *GetTabOrderList* to construct a list of child controls in tab order.

GetTabOrderList iterates through the windowed control's internal tab-order list, adding each of the controls to *List*, including any controls contained in those controls. The result is a list of all the controls and their owned controls, in tab order.

The *FindNextControl* method calls *GetTabOrderList* to build a complete list of the controls that *FindNextControl* uses to locate the next control in the tab order.

See also *FindNextControl* method, *TabOrder* property

GetTopParentHandle

GetTopParentHandle returns the window handle of the first window not associated with a VCL control that contains the windowed control.

```
function GetTopParentHandle: HWnd;
```

Description Use *GetTopParentHandle* to access the handle of the window that is hosting the windowed control. *GetTopParentHandle* walks up the containers of the windowed control until it finds a control whose *Parent* property is **nil**. If that topmost parent is contained in a non-VCL window (its *ParentWindow* is nonzero), *GetTopParentHandle* returns the non-VCL window. If the windowed control is not contained in a non-VCL window, *GetTopParentHandle* returns the windowed control's own window handle.

See also *GetParentHandle* method, *Handle* property, *Parent* property, *ParentWindow* property

HandleAllocated

HandleAllocated reports whether a window handle exists for the control.

```
function HandleAllocated: Boolean;
```

Description Query *HandleAllocated* to find out if the control has a window handle.

If a window handle exists, *HandleAllocated* returns *True*. If no window handle exists, *HandleAllocated* returns *False*. Testing the *Handle* property of a control directly causes a handle to be allocated if it does not already exist. Call the *HandleAllocated* method to determine whether a handle exists without allocating a handle as a side effect.

See also *Handle* property, *HandleNeeded* method

HandleNeeded

HandleNeeded creates a window handle for the control if it doesn't already exist.

```
procedure HandleNeeded;
```

Description Call *HandleNeeded* to create a window handle for the control.

If no window handle exists, *HandleNeeded* calls the *CreateHandle* method for the parent of the control before it creates a window handle for this control.

See also *Handle* property, *HandleAllocated* method

InsertControl

InsertControl inserts a control into the *Controls* array property of the windowed control.

```
procedure InsertControl(AControl: TControl);
```

Description Applications should not need to call *InsertControl* directly. Child controls are automatically inserted and removed when added or deleted at design time. At runtime, use the *Parent* property of the child control to insert it in the *Controls* array. If the child control is already the child of another windowed control, setting the *Parent* property ensures that the child is removed from the *Controls* of the original parent.

InsertControl makes the inserted control a child, and the containing control the parent. The *AControl* parameter is the child control that is inserted into the *Controls* array.

See also *Controls* property, *Parent* property, *RemoveControl* method

Invalidate

Invalidate tells Windows to repaint the control after other important Windows messages are handled.

```
procedure Invalidate; override;
```

Description Use *Invalidate* when the entire control needs to be repainted. When more than one region within the control needs repainting, *Invalidate* will cause the entire window to be repainted in a single pass, avoiding flicker caused by redundant repaints. There is no performance penalty for calling *Invalidate* multiple times before the control is actually repainted.

See also *Repaint* method, *Update* method

IsControlMouseMsg

IsControlMouseMsg returns *True* if a specified mouse message is directed to one of the child controls of the windowed control.

```
function IsControlMouseMsg(var Message: TWMMouse): Boolean;
```

Description Call *IsControlMouseMsg* to find out if a mouse message is directed to one of the control's child controls. Specify the mouse message as the value of the *Message* parameter.

Windows takes care of sending messages to windowed child controls, but for nonwindowed child controls, Windows sends the messages to the parent control, which must then determine which, if any, of its child controls should receive the message.

The *WndProc* method of a windowed control calls *IsControlMouseMsg* to process all mouse message sent to the windowed control.

See also *WndProc* method

KeyDown

KeyDown is the protected implementation method for a windowed control's *OnKeyDown* event.

```
type TShiftState = set of (ssShift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle, ssDouble);

procedure KeyDown(var Key: Word; Shift: TShiftState); dynamic;
```

Description The *KeyDown* method inherited from *TWinControl* does nothing except call any event handler attached to the *OnKeyDown* event. Override *KeyDown* to provide other responses in addition to the inherited event-handler call.

The *Key* parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for *VkKeyScan* and *VkKeyScanEx* in the Win32 Developer's Reference (WIN32.HLP).

A windowed control calls *KeyDown* in response to a key-down message (WM_KEYDOWN) from Windows. The actual sequence is that the message goes to a private message handler that calls the *DoKeyDown* method. If *DoKeyDown* determines that the control should, in fact, process the character, it decodes the parameters of the key-down message and passes the key code and shift-key state to *KeyDown* in the *Key* and *Shift* parameters, respectively.

Either *KeyDown* or the *OnKeyDown* event handler it calls can suppress further processing of a key by setting the *Key* parameter to zero.

The *Shift* parameter is a set of shift key and mouse combinations that includes these possible states:

State	Meaning
ssShift	The Shift key is held down.
ssAlt	The Alt key is held down.
ssCtrl	The Ctrl key is held down.
ssLeft	The left mouse button is held down.
ssMiddle	The middle mouse button is held down.
ssDouble	The mouse was double-clicked (not used for keystrokes).

See also *KeyUp* method, *OnKeyDown* event

KeyPress

KeyPress is the protected implementation method for a windowed control's *OnKeyPress* event.

```
procedure KeyPress(var Key: Char); dynamic;
```

Description The *KeyPress* method inherited from *TWinControl* does nothing except call any event handler attached to the *OnKeyPress* event. Override *KeyPress* to provide other responses in addition to the inherited event-handler call.

A windowed control calls *KeyPress* in response to a key-press message (WM_CHAR) from Windows. The actual sequence is that the message goes to a private message handler that calls the *DoKeyPress* method. If *DoKeyPress* determines that the control should, in fact, process the character, it decodes the parameters of the key-down message and passes the key code to *KeyPress* in the *Key* parameters, respectively.

Either *KeyPress* or the *OnKeyPress* event handler it calls can suppress further processing of a character by setting the *Key* parameter to zero.

Note The *Key* parameter is the character represented by the key that is pressed, not the Windows virtual key code.

See also *KeyDown* method, *KeyUp* method, *OnKeyPress* event

KeyUp

KeyUp is the protected implementation method for a windowed control's *OnKeyUp* event.

```
type TShiftState = set of (ssShift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle, ssDouble);

procedure KeyDown(var Key: Word; Shift: TShiftState); dynamic;
```

Description The *KeyUp* method simply calls the *OnKeyUp* event handler if it is assigned. Override *KeyUp* to provide other responses in addition to the inherited event-handler call.

A windowed control calls *KeyUp* in response to a key-up message (WM_KEYUP) from Windows. The actual sequence followed when a key is released is as follows:

1 The *DoKeyUp* method determines whether the control should, in fact, process the character.

2 If the character should be processed, DoKeyUp decodes the parameters of the key-up message and passes the key code and shift-key state to *KeyUp* in the *Key* and *Shift* parameters, respectively.

3 Either *KeyUp* or the *OnKeyUp* event handler it calls can suppress further processing of a key by setting the *Key* parameter to zero.

The *Key* parameter is the key on the keyboard. For non-alphanumeric keys, use WinAPI virtual key codes to determine the key pressed. For more information, search for *VkKeyScan* and *VkKeyScanEx* in the Win32 Developer's Reference (WIN32.HLP).

The *TShiftState* type is a set of shift key and mouse combinations that includes these possible states:

State	Meaning
ssShift	The Shift key is held down.
ssAlt	The Alt key is held down.
ssCtrl	The Ctrl key is held down.
ssLeft	The left mouse button is held down.
ssMiddle	The middle mouse button is held down.
ssDouble	The mouse was double-clicked (not used for keystrokes).

See also *KeyDown* method, *KeyPress* method, *OnKeyUp* event

MainWndProc

MainWndProc is the routine Windows calls when it has messages for the control.

```
procedure MainWndProc(var Message: TMessage);
```

Description Call *MainWndProc* to send a message to the control.

MainWndProc does not process or dispatch the messages itself, but rather calls the *WndProc* method to do that. *MainWndProc* provides an exception-handling block around *WndProc*, ensuring that if any unhandled exceptions occur during the processing of a message, the application's *HandleException* method is called to handle them.

See also *TApplication.HandleException* method, *WndProc* method

NotifyControls

NotifyControls sends a message to all the child windows of the windowed control.

```
procedure NotifyControls(Msg: Word);
```

Description The *NotifyControls* method sends a message with the message ID passed in the *Msg* parameter to all the controls in the windowed control's *Controls* array property. Delphi uses *NotifyControls* to update all the controls in a form of such occurrences as changes in the parent color or font.

NotifyControls constructs a generic message record, filling its message ID field with the value of *Msg* and setting its parameter and result fields to zeros, then calls the *Broadcast* method to send the message to all the child controls.

See also *Broadcast* method, *Controls* property

PaintControls

PaintControls paints each of the child controls in a windowed control using the specified device context.

```
procedure PaintControls(DC: HDC; First: TControl);
```

Description Call *PaintControls* to paint all the child controls of this control. Pass the device context as the value of the *DC* parameter. The *First* parameter indicates the starting point in the windowed control's child-control list to paint. If *First* is **nil** or does not indicate one of the child controls, *PaintControls* paints all the child controls.

The *Repaint* method calls *PaintControls* for the control's parent, passing the control in *First* to repaint the control and any controls it might intersect. *PaintHandler* also calls *PaintControls* after calling *PaintWindow*.

See also *PaintHandler* method, *PaintWindow* method, *Repaint* method

PaintHandler

PaintHandler is a handler that responds to WM_PAINT messages.

```
procedure PaintHandler(var Message: TWMPaint);
```

Description *PaintHandler* calls the *BeginPaint* and *EndPaint* API functions, and between them paints the control's background by calling *PaintWindow* and any child controls by calling *PaintControls*.

See also *PaintControls* method, *PaintWindow* method

W

PaintTo

PaintTo draws the windowed control on a device context.

```
procedure PaintTo(DC: HDC; X, Y: Integer);
```

Description Call *PaintTo* to draw the control on a device context. Specify the device context as the value of the *DC* parameter and specify the X and Y coordinates on the device context where the top-left corner of the windowed control is to be drawn. *PaintTo* first erases the background of the device context and then paints the control.

PaintTo is useful for drawing an image of the control into a bitmap DC.

See also *PaintWindow* method

PaintWindow

PaintWindow renders the image of a windowed control.

```
procedure PaintWindow(DC: HDC); virtual;
```

Description Call *PaintWindow* to repaint the control. *PaintWindow* sends a WM_PAINT message to the windowed control's *DefaultHandler* method, setting the message record's *WParam* field to the value passed in *DC* and the other parameter and result fields to zeros.

See also *DefaultHandler* method, *PaintTo* method

PaletteChanged

PaletteChanged responds to changes in the system's palette by realizing the control's palette and the palette for each child control.

```
function PaletteChanged(Foreground: Boolean): Boolean; dynamic;
```

Description Override *PaletteChanged* to change how the control responds to Windows notifications that the system palette is changing. *TWinControl* responds to these notifications by trying to realize the windowed control's palette, if any, into the current device context, and then passing on the *PaletteChanged* notification to each of the child controls in turn. If the change in the system palette causes the actual value of the control's palette to change, *PaletteChanged* will invalidate the control so that it can repaint with the new palette.

Windows paints the active window with a foreground palette, while other windows are forced to use background palettes. Background palettes are approximate matches to the colors specified by the control's logical palette, given the limitations imposed by implementing the foreground palette. Windows only allows a single foreground palette.

PaletteChanged allows the control to obtain a new realization of its palette when the system palette changes. When *ForeGround* is *True*, the form has been activated and the control is specifying the new foreground palette. When *ForeGround* is *False*, another application or form has changed the foreground palette, and controls that are sensitive to the available palette should realize new background palettes to best match their logical palettes.

If the control does not have a logical palette to be realized into the current device context, *GetPalette* will return a handle of 0, and *PaletteChanged* will not try to realize a palette for the windowed control. Instead, it simply passes the notification on to the child controls. If the windowed control or any of its child controls realize a palette into the current device context in response to *PaletteChanged*, this method returns *True*.

Override *PaletteChanged* to change the way in which controls are given an opportunity to realize a new palette when the windows system palette changes. For example, override *PaletteChanged* to change the order in which child controls realize their palettes, giving a particular control the foreground palette when the form is activated, or to pass palette change notification to custom controls implemented outside of Delphi, such as in DLLs. Override *GetPalette* instead to affect whether the windowed control actually has a palette to realize, or to change the value of the control's logical palette.

Note *PaletteChanged* is called only when the runtime video mode requires palette support, such as for 256 color mode, but not 16 million color mode.

See also *GetDeviceContext* method, *GetPalette* method

ReadState

ReadState prepares the control for having its properties assigned values read from a stream.

```
procedure ReadState(Reader: TReader); override;
```

Description Override *ReadState* to change the preparations the control makes as it readies itself to have its properties values assigned from a stream. For example, the windowed control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Be sure to include a call to the inherited method when overriding *ReadState*.

ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited *ReadState* method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its *Ctl3D* property and its visibility with that of its parent control.

See also *Parent* property, *TControl.ReadState* method, *TReader* object

Realign

Realign forces the windowed control to realign the controls within it.

```
procedure Realign;
```

Description Call *Realign* to realign all the child controls of this control after modifying something that affects their alignment.

If all of the child controls of this control have their *Align* properties set to *alNone*, *Realign* has no effect.

The *EnableAlign* method calls *Realign* when its reference count reaches zero.

See also *Align* property, *DisableAlign* method, *EnabledAlign* method

RecreateWnd

RecreateWnd destroys the window associated with a windowed control so that another can be generated to reflect the control's current state.

```
procedure RecreateWnd;
```

Description Call *RecreateWnd* to recreate the control. *RecreateWnd* destroys the window used by the control, setting the window handle to NULL. The next time the window handle is needed, such as when the control needs to be painted, a new window is created that reflect's the properties of the control.

Windows does not reflect certain kinds of changes to a control without destroying the window and recreating it. For example, to reflect a change in the border style of an edit box, the window for the control must be recreated.

See also *DestroyHandle* method, *Handle* property, *HandleNeeded* method

RemoveControl

RemoveControl removes a specified control from the *Controls* array.

```
procedure RemoveControl(AControl: TControl);
```

Description Applications should not need to call *RemoveControl* directly. Child controls are automatically inserted and removed when added or deleted at design time. At runtime, use the *Parent* property of the child control to remove it from the *Controls* array.

RemoveControl removes a child control from the *Controls* property. The result is that this control is no longer the parent of the child specified by the *AControl* parameter.

See also *Controls* property, *InsertControl* method, *Parent* property

Repaint

Repaint repaints the windowed control's image on the screen.

```
procedure Repaint; override;
```

Description Call *Repaint* to repaint the control.

The *Repaint* method calls the *Invalidate* method and then the *Update* method to repaint the control.

See also *Invalidate* method, *Update* method

ResetIme

ResetIme restores the input method editor (IME) that was active when the application started.

```
procedure ResetIme;
```

Description For controls that specify the *ImeName* property, call *ResetIme* to revert to the IME that was active when the application started. If the IME was deactivated by setting the *IMEMode* property to *imDisable*, *ResetIme* restarts the default IME.

Applications cannot call *ResetIme*. *ResetIme* is called automatically when the control loses input focus. It is exposed as a protected method so that descendant objects can restore the default IME at other times.

See also *ImeMode* property, *ImeName* property, *SetIme* method, *TScreen.DefaultIme* property

ResetImeComposition

ResetImeComposition directs the composition window of the input method editor (IME) to perform a specific action.

```
function ResetImeComposition(Action: DWORD): Boolean;
```

Description Call *ResetImeComposition* to direct what the composition window does with the input typed by the user. The composition window is the window displayed by the IME that provides feedback to the user about what keystrokes have been typed, and how they are converted into Asian characters.

The *Action* parameter indicates what the composition window should do with its current contents. *Action* can have one of the following values:

Value	Meaning
CPS_CANCEL	Clear the composition string and set the status to no composition string.
CPS_COMPLETE	Set the composition string as the result string.
CPS_CONVERT	Convert the composition string.
CPS_REVERT	Cancel the current composition string and revert to the unconverted string.

ResetImeComposition returns *True* if the composition window performs the requested action.

See also *ImeMode* property, *ImeName* property, *SetImeCompositionWindow* method

W

ScaleBy

ScaleBy rescales a windowed control and any of its children to a percentage of their former size.

```
procedure ScaleBy(M, D: Integer);
```

Description Call *ScaleBy* to rescale the control and all its children. *ScaleBy* modifies the *Height* and *Width* properties of the controls, but it does not change the *Top* and *Left* properties; therefore, the size of the controls changes, but not the coordinates of the top-left corners of the controls.

The *M* and *D* parameters determine the percentage by which the control and its children are scaled. The *M* parameter is the multiplier and the *D* parameter is the divisor. For example, to change a control to be 75% of its original size, specify the value of *M* as 75 and the value of *D* as 100 (75/100). Alternately, specify the value of *M* as 3 and the value of *D* as 4 (3/4). Both fractions are equal and result in the control being scaled by 75%.

To make the control 33% larger than its previous size, specify the value of *M* as 133 and the value of *D* as 100 (133/100), or specify the value of *M* as 4 and the value of *D* as 3 (4/3); the fraction 133/100 is approximately equal to 4/3.

Note Because control sizes and coordinates are integers, scaling them by the *M*/*D* ratio incurs some degree of roundoff error. Calling *ScaleBy* repeatedly may mean that at some point the control disappears (when the roundoff errors make the control smaller). When rescaling controls repeatedly, consider destroying and reloading the form after several rescalings so that the coordinates are rescaled without most of the accumulating roundoff error.

See also *ChangeScale* method, *ScaleControls* method

ScaleControls

ScaleControls rescales the child controls of a windowed control without rescaling the control itself.

```
procedure ScaleControls(M, D: Integer);
```

Description Call *ScaleControls* to rescale only the children of the control, while leaving the control itself the same size.

ScaleControls calls the *ChangeScale* method of each child control, specifying the percentage to scale by through the *M* and *D* parameters. *M* is the multiplier and *D* is the divisor. For example, to change the child controls to be 75% of their original size, specify the value of *M* as 75 and the value of *D* as 100 (75/100). Alternately, specify the value of *M* as 3 and the value *D* as 4 (3/4). Both fractions are equal and result in the controls being scaled by 75%.

Note Because control sizes and coordinates are integers, scaling them by the *M*/*D* ratio incurs some degree of roundoff error. Calling *ScaleBy* repeatedly means that at some point the control may disappear (when the roundoff errors make the control smaller), or creep off the form (when the roundoff errors

move the location of control). When rescaling controls repeatedly, consider destroying and reloading the form after several rescalings so that the coordinates are rescaled without most of the accumulating roundoff error.

See also *ChangeScale* method, *ScaleBy* method

ScrollBy

ScrollBy scrolls the contents of a windowed control.

```
procedure ScrollBy(DeltaX, DeltaY: Integer);
```

Description Call *ScrollBy* to scroll the contents within the control. While *ScrollBy* can be used for any windowed control, it makes the most sense to use it for descendants of *TScrollingWinControl*.

Applications seldom need to call the *ScrollBy* method unless they implement their own scrolling interface rather than relying on a scroll bar.

The *DeltaX* parameter is the change in pixels along the X axis. A positive *DeltaX* value scrolls the contents to the right; a negative value scrolls the contents to the left. The *DeltaY* parameter is the change in pixels along the Y axis. A positive *DeltaY* value scrolls the contents down; a negative value scrolls the contents up.

See also *TScrollingWinControl.HorzScrollBar* property, *TScrollingWinControl.ScrollInView* method, *TScrollingWinControl.VertScrollBar* property

SelectFirst

SelectFirst locates the first child selectable control in the tab order within the windowed control.

```
procedure SelectFirst;
```

Description Call *SelectFirst* to find the first child that can be selected within the control's tab order. *SelectFirst* calls the *FindNextControl* method, passing **nil** as the current control. Once the first selectable control is found, *SelectFirst* makes the control the active control on the form.

See also *FindNextControl* method, *SelectNext* method, *TForm.ActiveControl* property

SelectNext

SelectNext moves the input focus from the current child control to the next one in the tab order.

```
procedure SelectNext(CurControl: TWinControl; GoForward, CheckTabStop: Boolean);
```

Description Call *SelectNext* to find the next child control in the control's tab order. Specify the current control from which to begin the search as the value of the *CurControl* parameter. The *GoForward* parameter controls the direction of the

search. If *GoForward* is *True*, *FindNextControl* searches forward through the child controls in tab order. If *GoForward* is *False*, *FindNextControl* searches backward through the controls.

The *CheckTabStop* parameter controls whether the control *SelectNext* finds must be a tab stop. If *CheckTabStop* is *True*, the returned control must have its *TabStop* property set to *True*, or the search for the next control continues.

If *SelectNext* doesn't locate an appropriate control (as defined by the *GoForward* and the *CheckTabStop* parameters), the focus remains with *CurControl*.

See also *SelectFirst* method

SetBounds

SetBounds sets the windowed control's boundary properties all at once.

```
procedure Setbounds(ALeft, ATop, AWidth, AHeight: Integer); override;
```

Description *Use SetBounds* to change all of the control's boundary properties at once. The same effect can be achieved by setting the *Left*, *Top*, *Width*, and *Height* properties separately. By setting all four properties at once, *SetBounds* ensures that the control will not repaint between changes

Specify the values for the *Left*, *Top*, *Width*, and *Height* properties as the value of the *ALeft*, *ATop*, *AWidth*, and *AHeight* parameters, respectively.

See also *Height* property, *Left* property, *Top* property, *Width* property

SetChildOrder

SetChildOrder changes the order in which the child component appears in the list of child components returned by the *GetChildren* method.

```
procedure SetChildOrder(Child: TComponent; Order: Integer); override;
```

Description Use *SetChildOrder* when you want to change the order in which child objects are streamed in. Specify the child component whose order you want to change as the value of the *Child* parameter. Indicate the position you want the component to be in as the value of the *Order* parameter.

When *SetChildOrder* is called, items previously below the child's old position move up, and those below the new position move down.

See also *GetChildren* method

SetFocus

SetFocus gives the input focus to the control.

```
procedure SetFocus; virtual;
```

Description Use *SetFocus* to change the focus to another control.

See also *CanFocus method*

SetIme

SetIme activates an input method editor (IME) using the values specified by the *ImeMode* and *ImeName* properties.

```
procedure SetIme;
```

Description Applications cannot call *SetIme*. *SetIme* is called automatically when the control receives input focus. It is exposed as a protected method so that descendant objects can cause the *ImeMode* and *ImeName* properties to take effect at other times.

See also *ImeMode* property, *ImeName* property

SetImeCompositionWindow

SetImeCompositionWindow sets the position and font of the composition window.

```
function SetImeCompositionWindow( Font: TFont; XPos, YPos: Integer): Boolean;
```

Description Call *SetImeComposition* to change the position or font of the composition window. The composition window is the window displayed by the input method editor (IME) that provides feedback to the user about what keystrokes have been typed, and how they are converted into Asian characters.

The *Font* parameter describes the font that the composition window should use. Setting the *Font* parameter to **nil** repositions the composition window without changing the font. *XPos* and *YPos* specify the desired global coordinates for the composition window.

SetImeCompositionWindow returns *True* if the font and position are successfully assigned.

See also *ImeMode* property, *ImeName* property, *TFont* object

SetZOrder

SetZOrder moves the control within the parent's control list to either the top or bottom of the windowed controls; or, if the control has no parent, the control becomes either the top or bottom window on the screen.

```
procedure SetZOrder(TopMost: Boolean); override;
```

Description Use *SetZOrder* to move rearrange overlapping controls within a parent control or overlapping windows on the screen. If the *TopMost* parameter is *True*, the control becomes the top control; otherwise, it becomes the bottom control.

Windowed controls always stack on top of non-windowed controls. Thus, if the *Parent* contains any non-windowed controls, calling *SetZOrder* with a *TopMost* value of *False* will still keep the windowed control above all the non-windowed controls.

After changing the order, *SetZOrder* invalidates the control to ensure repainting to reflect the new order.

See also *BringToFront* method, *Invalidate* method, *Parent* property, *SendToBack* method

ShowControl

ShowControl ensures that a specified child control is visible.

```
procedure ShowControl(AControl: TControl); virtual;
```

Description Call *ShowControl* to display a child control. Specify the control you want to ensure will be visible as the value of the *AControl* parameter.

The *TWinControl* method *ShowControl* simply calls the *ShowControl* method of the windowed control's parent. Derived objects override *ShowControl* to perform whatever actions are necessary to allow the indicated child object to be visible. Possible actions include scrolling the particular child object into view, or changing to the appropriate notebook page.

ShowControl is a protected method used in the implementation of the *Show* method.

See also *Show* method

Update

Update processes any pending paint messages immediately.

```
procedure Update; override;
```

Description Call *Update* to force the control to be repainted before any more, possibly time-consuming, processing takes place. Use *Update* to provide immediate feedback to the user that cannot wait for the Windows paint message to arrive. *Update* does not invalidate the control, but simply forces a repaint of any regions that have already been invalidated. Call *Repaint* instead if you need to invalidate the control as well.

Update calls the Windows API *UpdateWindow* function, which processes any pending paint messages. For more information, see the Win32 Developer's Reference (WIN32.HLP).

See also *Invalidate* method, *Repaint* method

UpdateControlState

UpdateControlState shows a windowed control, causing all parent objects to make any necessary adjustments.

```
procedure UpdateControlState;
```

Description *UpdateControlState* is used internally by Delphi to show controls and create handles for controls at the time they are displayed. *UpdateControlState* calls the *Show* method to show the windowed control. To help ensure that the windowed control will be visible, it creates the window handle for the object or any parent objects that do not yet have one, and calls the *ShowControl* method of all parent controls to ensure that the control is in view.

UpdateControlState shows the control only if its parent is showing and the control's *Visible* property is *True*.

See also *Handle* property, *Show* method, *ShowControl* method, *Showing* property

WndProc

WndProc provides specific message responses for the control.

```
procedure WndProc(var Message: TMessage); override;
```

Description Override *WndProc* to change the initial Windows message handler for the control. The *WindowProc* property is initialized to point to the *WndProc* method.

WndProc for *TWinControl* overrides the inherited method to define responses for focus, mouse, and keyboard messages. It sends all others to its inherited *WndProc*.

When overriding *WndProc* to provide specialized responses to messages, call the inherited *WndProc* at the end to dispatch any unhandled messages.

See also *DefWndProc* property, *MainWndProc* property, *TControl.WndProc* method, *WindowProc* property

TWordField

Unit: DB

TObject • TPersistent • TComponent • TField • TNumericField • TIntegerField • TWordField

A *TWordField* object represents an unsigned 16-bit integer field in a dataset.

Description Word fields can hold values in the range 0 to 65535, and are used to hold values that are unsigned 16-bit whole numbers.

TWordField differs from its immediate ancestor *TIntegerField* only in the data type it represents. Except for the constructor and the *DataSize* property, all of the properties, methods, and events are the same as those of *TIntegerField*, and provide functionality that is useful for managing the value and properties of an integral numeric field in a database.

See also *TDataSet object*, *TSmallIntField* object

Properties

The following table lists the properties of *TWordField*:

Property	Visibility	Default	Documented in
Alignment	published	taRightJustify	TField
AsBoolean	public		TField
AsCurrency	public		TField
AsDateTime	public		TField
AsFloat	public		TIntegerField
AsInteger	public		TIntegerField
AsString	public		TIntegerField
AsVariant	public		TIntegerField
AttributeSet	public		TField
Calculated	public	False	TField
CanModify	public, read-only		TField
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
ConstraintErrorMessage	published		TField
CurValue	public, read-only		TField
CustomConstraint	published		TField
DataSet	public		TField
DataSize	public, read-only		TWordField
DataType	public, read-only		TField
DefaultExpression	published		TField
DesignInfo	public		TComponent
DisplayFormat	published		TNumericField
DisplayLabel	published		TField
DisplayName	public, read-only		TField
DisplayText	public, read-only		TField
DisplayWidth	published		TField
EditFormat	published		TNumericField
EditMask	public		TField
EditMaskPtr	public, read-only		TField
FieldKind	published	fkData	TField
FieldName	published		TField
FieldNo	public, read-only		TField
ImportedConstraint	published		TField
Index	published		TField

Property	Visibility	Default	Documented in
IsIndexField	public, read-only		TField
IsNull	public, read-only		TField
KeyFields	published		TField
Lookup	public		TField
LookupCache	published	False	TField
LookupDataSet	published		TField
LookupKeyFields	published		TField
LookupList	public, read-only		TField
LookupResultField	published		TField
MaxValue	published	0	TIntegerField
MinValue	published	0	TIntegerField
Name	published		TComponent
NewValue	public		TField
Offset	public, read-only		TField
OldValue	public, read-only		TField
Origin	published		TField
Owner	public, read-only		TComponent
ReadOnly	published	False	TField
Required	published	False	TField
Size	public		TField
Tag	published	0	TComponent
Text	public		TField
ValidChars	public		TField
Value	public		TIntegerField
VCLComObject	public		TComponent
Visible	published	True	TField

DataSize

DataSize is the amount of memory that is needed to store the value of a word field in its native database format.

```
property DataSize: Word;
```

Description Check *DataSize* to determine the number of bytes required to store a word field component's value. Use *DataSize* to determine the size needed for a buffer when working with the *GetData* and *SetData* methods.

For word fields, *DataSize* is always 2. This is the size of unsigned word values in the database tables, not the size of the LongInt data type used to store and manipulate values by *TWordField*.

See also *GetData* method, *SetData* method, *Value* property

Events

The following table lists the events of *TWordField*:

Event	Visibility	Documented in
OnChange	published	TField
OnGetText	published	TField
OnSetText	published	TField
OnValidate	published	TField

Methods

The following table lists the methods of *TWordField*:

Method	Visibility	Documented in
Assign	public	TField
AssignValue	public	TField
ClassInfo	public	TObject
ClassName	public	TObject
ClassNameIs	public	TObject
ClassParent	public	TObject
ClassType	public	TObject
CleanupInstance	public	TObject
Clear	public	TField
Create	public	TWordField
DefaultHandler	public	TObject
Destroy	public	TField
DestroyComponents	public	TComponent
Destroying	public	TComponent
Dispatch	public	TObject
FieldAddress	public	TObject
FindComponent	public	TComponent
FocusControl	public	TField
Free	public	TObject
FreeInstance	public	TObject
FreeNotification	public	TComponent
FreeOnRelease	public	TComponent
GetData	public	TField
GetInterface	public	TObject
GetInterfaceEntry	public	TObject
GetInterfaceTable	public	TObject
GetNamePath	public	TPersistent
GetParentComponent	public	TComponent

Method	Visibility	Documented in
HasParent	public	TComponent
InheritsFrom	public	TObject
InitInstance	public	TObject
InsertComponent	public	TComponent
InstanceSize	public	TObject
IsBlob	public	TField
IsValidChar	public	TField
MethodAddress	public	TObject
MethodName	public	TObject
NewInstance	public	TObject
RefreshLookupList	public	TField
RemoveComponent	public	TComponent
SafeCallException	public	TComponent
SetData	public	TField
SetFieldType	public	TField
Validate	public	TField

Create

The *Create* method creates an instance of *TWordField*.

```
constructor Create(AOwner: TComponent); override;
```

Description It is seldom necessary to call *Create* directly, because a word field component is instantiated automatically for all fields that contain only word values in a dataset.

After calling the inherited constructor, *Create* sets

- *DataType* to *ftWord*
- *MinValue* to 0
- *MaxValue* to 65535

See also *DataType property*, *MaxValue property*, *MinValue property*, *TField.Create* method

Index

Symbols

+ button 650
– button 650

Numerics

2-D controls 1635, 1638
3-D controls 1635, 1638

A

aaDoubleClick constant 1267
aaGetFocus constant 1267
aaManual constant 1267
AbortOnKeyViol property
  TBatchMove 51
AbortOnProblem property
  TBatchMove 52
About boxes 1291
accelerator tables 1185
accelerators 456, 1130, 1242
  creating 461
  labels 484
  static text controls 634
  testing for 1235
access keys *See* accelerators
AcquireLayoutLock method
  TCustomDBGrid 332
  TDSTableProducer 1021
action items 674, 677, 678, 1629
  See also Web server applications
Action property
  TCustomWebDispatcher 674
ActionByName method
  TCustomWebDispatcher 677
Actions property
  TCustomWebDispatcher 674
Activate method
  TCustomForm 411
ActivateHint method
  THintWindow 1128
ActivateHintData method
  THintWindow 1129
ActivateWebModule method
  TWebApplication 1623
active application 22, 29
  bringing to front 37
active control, returning 1419
active datasets 703
active form 411, 1419, 1420, 1423
  changing 411
  deactivating 405, 415
  host applications 3
  setting 393, 403, 411
  testing for 392, 393
Active property
  TAnimate 10
  TApplication 22
  TCustomForm 392
  TCustomSocket 620
  TDataSet 703
  TDecisionCube 932
  TSession 1455
ActiveBuffer method
  TDataSet 724
ActiveChanged method
  TCustomForm 411
ActiveControl property
  TCustomForm 393
  TScreen 1419
ActiveCount property
  TWebApplication 1620
ActiveCustomForm property
  TScreen 1419
ActiveForm objects 3 to 8
  borders 4
  creating 7
ActiveForm property
  TScreen 1420
ActiveMDIChild property
  TCustomForm 393
ActiveOleControl property
  TCustomForm 393
ActivePage property
  TPageControl 1304
ActiveX applications
  active form 3
ActiveX controls
  copying properties 1355
  editing 1355
  property pages 1349
  references 1351
Add method
  See also Insert method
  TCustomImageList 468
  TMenuItem 1244
AddAlias method
  TSession 1464
AddButtonClick method
  TPasswordDialog 1326
AddDataSet method
  TCustomRemoteServer 587
AddDriver method
  TSession 1465
AddIcon method
  TCustomImageList 469
AddImages method
  TCustomImageList 469
AddIndex method
  TClientDataSet 167
  TTable 1548
adding components 211
AddMasked method
  TCustomImageList 469
AddPassword method
  TSession 1466
_AddRef method
  TComponent 217
Address property
  TCustomSocket 621
AddStandardAlias method
  TSession 1466
AdjustSize method
  TCustomGridp 438
afbNone constant 5
afbRaised constant 5
afbSingle constant 5
afbSunken constant 5
AfterCancel event
  TDataSet 713
AfterClose event
  TCustomDataStore 309
  TDataSet 713
AfterConnect event
  TCustomRemoteServer 585
AfterDelete event
  TDataSet 714
AfterDisconnect event
  TCustomRemoteServer 585
AfterDispatch event
  TCustomWebDispatcher 675
AfterEdit event
  TDataSet 714
AfterInsert event
  TDataSet 714
AfterOpen event
  TCustomDataStore 310
  TDataSet 715
AfterPost event
  TDataSet 715

AfterScroll event
TDataSet class 715
alBottom constant 219
alClient constant 220
aliases
database engine 682, 1470, 1471
adding 1464, 1466
changing 1474
removing 1468
testing for 1473
database sessions 688, 1456
AliasName property
TDatabase 682
Align property
TControl 219
TCoolBar 265
AlignButton property
TCustomUpDown 664
AlignControls method
TCustomPanel 570
TCustomTabControl 644
TScrollingWinControl 1448
TWinControl 1647
aligning
captions
check boxes 276
panels 564
radio buttons 1380
controls 219, 570, 644, 1647, 1654, 1657, 1667
data 1035
text 482, 549, 556, 633
alignment constants 219
edit controls 549
field objects 1035
labels 482
menus 1332
panels 564
radio buttons 1380
static text controls 633
table producers 1014
up-down controls 664
Alignment property
TCustomCheckBox 276
TCustomLabel 482
TCustomMemo 549
TCustomPanel 564
TCustomStaticText 633
TField 1035
TPopupMenu 1332
TRadioButton 1380
alLeft constant 220
allocating memory
data stores 304, 307, 308, 315
AllocBy property
TCustomImageList 463
TCustomListView 509
AllowActiveDoc property
TOleContainer 1267
AllowAllUp property
TSpeedButton 1486
TToolButton 1581
AllowDelete property
TDBCtrlGrid 793
AllowGrayed property
TCheckListBox 147
TCustomCheckBox 277
AllowInPlace property
TOleContainer 1267
AllowInsert property
TDBCtrlGrid 794
AllowTesting property
TApplication 22
alNone constant 219
AlphaSort method
TCustomListView 523
TCustomTreeView 658
alRight constant 220
alTop constant 219
ancestors 213, 1257, 1262
animation controls 8 to 20
centering frames 10
closing 16
color settings 15
common AVI clips 11
counting frames 11
creating 19
events 16, 17
file names 11
names 14
opening 12, 16
playing 10, 19
repeating sequences 13
resetting 20
resource IDs 13
size 10, 12
starting 14, 16
stopping 15, 17, 20
timers 15
ANSI characters 380
conversions 1524
Append method
TDataSet 725
AppendData method
TClientDataSet 167
appending records 167, 725
AppendRecord method
TDataSet 725
application servers
connections 162, 588, 1389, 1391, 1392
dropping remote 590, 1391
data providers 160, 572, 589, 1356
handling data requests 574, 576
names 1389, 1390
updating data 575
Application variable 1619
applications 21, 1418
activating 22, 29
closing 28, 45
creating 38
destroying instances 39
event handling 29 to 34
frontmost 37
getting active form 1419, 1420, 1423
main window 26, 27
maximizing 33, 44
message processing 32, 39, 41, 44, 46
minimizing 26, 28, 33, 43
running 45
setting defaults 28
tracking progress 1343
Web sites 138, 1160, 1619
Apply method
TUpdateSQL 1609
ApplyFilePath method
TFileListBox 1072
ApplyRange method
TClientDataSet 168
TTable 1548
ApplyUpdates method
TBDEDataSet 83
TClientDataSet 168
TCustomProvider 575
TDatabase 693
TProvider 1360
AppServer property
TRemoteServer 1389
arAlignBottom constant 524
arAlignLeft constant 524
arAlignRight constant 524
arAlignTop constant 524
archives 1068
arDefault constant 524
Arrange method
TCustomListView 524
ArrangeIcons method
TForm 1104
arrays

components 199
data grids 348
databases 1458
datasets 684, 707
decision grids 360, 371
field components 321
list views 530
MDI forms 397
menus 1241, 1246
page sets 1305
radio buttons 579, 580, 887
tab sheets 1560
ArrowKeys property
TCustomUpDown 665
arSnapToGrid constant 524
AsBoolean property
TBooleanField 121
TField 1035
TStringField 1521
ASCII characters 59, 114, 1524
ASCII tables 1466, 1546
AsCurrency property
TBCDField 64
TField 1035
AsDateTime property
TDateTimeField 762
TField 1036
TStringField 1522
AsFloat property
TBCDField 64
TDateTimeField 763
TField 1036
TFloatField 1088
TIntegerField 1155
TStringField 1522
Asian character sets 1424, 1426, 1636, 1637
composition window 1669, 1673
AsInteger property
TBCDField 65
TField 1036
TFloatField 1088
TIntegerField 1156
TStringField 1522
Assign method
TBlobField 116
TCustomImageList 469
TField 1059
TPersistent 1329
assigning values
binary fields 97, 98
binary-coded decimals 64, 65, 66, 67
BLOBs 112, 114, 116
Boolean fields 121, 122, 123, 1035
currency fields 1035
date/time fields 762, 763, 764, 766, 1036
field objects 1053, 1060
floating-point numbers 1036, 1088, 1089, 1090, 1091
integers 1036, 1155, 1156, 1157, 1158
radio buttons 886, 887, 888
strings 1037, 1521, 1522, 1523, 1524
variants 1037
assignments 199
AssignTo method
TPersistent 1329
AssignValue method
TField 1060
Associate property
TCustomUpDown 665
AsString property
TBCDField 65
TBlobField 112
TBooleanField 121
TDateTimeField 763
TField 1037
TFloatField 1089
TIntegerField 1156
TStringField 1523
AsVariant property
TBCDField 66
TBinaryField 97
TBlobField 112
TBooleanField 122
TDateTimeField 764
TField 1037
TFloatField 1089
TIntegerField 1156
TStringField 1523
asynchronous client sockets 182
attaching to databases 680, 696
DDE applications 906, 917, 921, 927
attaching to remote servers 162, 583, 588, 1388, 1389, 1391, 1392
attaching to Windows sockets 183, 620, 629, 630
opening connections 630
server types 607, 616, 619
AttributeSet property
TField 1037
Audio Video Interleaved clips *See* AVI clips
AutoActivate property
TOleContainer 1267
AutoButtonSet method
TMediaPlayer 1212
AutoCalcFields property
TDataSet 704
AutoDisplay property
TDBImage 831
TDBMemo 869
TDBRichEdit 894
AutoEdit property
TDataSource 750
AutoEnable property
TMediaPlayer 1194
autoincrementing field type 46
automation servers *See* OLE Automation
AutoMerge property
TMainMenu 1183
AutoOpen property
TMediaPlayer 1195
AutoPopup property
TPopupMenu 1333
AutoRewind property
TMediaPlayer 1196
AutoScroll property
TScrollingWinControl 1445
AutoScrollInView method
TScrollingWinControl 1448
AutoSelect property
TCustomEdit 378
AutoSessionName property
TSession 1456
AutoSize property
TAnimate 10
TCoolBar 265
TCustomEdit 378
TCustomHotKey 458
TCustomLabel 483
TDBText 902
TImage 1135
TToolBar 1573
Autosize property
TCustomStaticText 633
AutoUnload property
TReport 1397
AutoVerbMenu property
TOleContainer 1268
AVI clips 8
AXBorderStyle property
TActiveForm 4

B

Back method
TMediaPlayer 1213

background colors 223, 1633
animation frames 15
data grids 797
decision grids 360, 361, 364
grids 428
images 463
BandBorderStyle property
TCoolBar 266
Bands editor 266
Bands property
TCoolBar 266
batAppend constant 54, 56, 59, 1549
batAppendUpdate constant 54, 56, 59, 1549
batch processing 50, 1549
abnormal termination 51, 52, 54, 55, 57
backing up previous records 53
character conversions 59
column mappings 55
counting records 52, 53, 54, 57, 58
data sources 58
destinations 53
initiating 60, 61
modes 56
terminating 61
BatchMove method
TTable 1549
batCopy constant 54, 56, 59, 1549
batDelete constant 54, 56, 59, 1549
batUpdate constant 54, 56, 59, 1549
BCD fields 61
assigning values 64, 65, 66, 67
creating 69
formatting 66
storing values 67
BDE *See* Borland Database Engine
BeforeCancel event
TDataSet 716
BeforeClick event
TDBNavigator 880
BeforeClose event
TCustomDataStore 310
TDataSet 716
BeforeConnect event
TCustomRemoteServer 585
BeforeDelete event
TDataSet 716
BeforeDisconnect event
TCustomRemoteServer 586
BeforeDispatch event
TCustomWebDispatcher 676
BeforeEdit event
TDataSet 717
BeforeInsert event
TDataSet 717
BeforeOpen event
TCustomDataStore 310
TDataSet 717
BeforePost event
TDataSet 718
BeforeScroll event
TDataSet class 718
BeginDrag method
TControl 244
TCustomImageList 470
BeginLayout method
TCustomDBGrid 332
TDSTableProducer 1021
BeginUpdate method
TCustomDBGrid 332
TDSTableProducer 1021
beveled panels 91, 1494
creating 95
shapes 92
size 565
styles 93, 564, 565
Beveled property
TSplitter class 1494
BevelInner property
TCustomPanel 564
BevelOuter property
TCustomPanel 565
BevelWidth property
TCustomPanel 565
biHelp constant 394
biMaximize constant 394
biMinimize constant 394
binary data 95
binary fields 95
assigning values 97, 98
creating 99
binary large objects *See* BLOB fields; BLOBs
binary-coded decimals 61
assigning values 64, 65, 66, 67
formatting 66
storing 67
BinData property
TCustomDataStore 304
biSystemMenu constant 394
bitmap buttons 100
See also bitmaps
adding captions 105
creating/destroying 109
event handling 108
getting images 102, 105
laying out images 104
multiple images 105
styles 106
bitmap frames 8
Bitmap property
TCoolBar 266
bitmaps
See also bitmap buttons
image lists 469, 473, 474, 475, 476, 477, 478
tree views 648
state images 651
bkAbort constant 103
bkAll constant 104
bkCancel constant 103
bkClose constant 103
BkColor property
TCustomImageList 463
bkCustom constant 103
bkHelp constant 103
bkIgnore constant 104
bkNo constant 103
bkOK constant 103
bkRetry constant 103
bkYes constant 103
blank values 1060
testing for 113, 1048
BlendColor property
TCustomImageList 463
blGlyphBottom constant 104, 1489
blGlyphLeft constant 104, 1489
blGlyphRight constant 104, 1489
blGlyphTop constant 104, 1489
BLOB fields 109
assigning values 112, 114, 116
cached updates 1052
changing values 114
character conversions 114
clearing values 117
creating 117
displaying images 829
exceptions 119
loading objects 117, 869, 875
saving contents 118
testing for 1062
testing for nulls 113

type constants 118
BLOBs 829
See also BLOB fields
caching 71
displaying 71, 831
reading 117
typecasting 113
BlobType property
TBlobField 113
BOF property
TDataSet 704
Bookmark property
TDataSet 705
bookmarks 705, 735
comparing 86, 170, 728
moving to 736
testing validity 84, 168
validating 726
BookmarkValid method
TBDEDataSet 84
TClientDataSet 168
TDataSet class 726
Boolean fields 119
assigning values 121, 122, 123, 1035
creating 125
storing values 122
Boolean values 121, 1035
conversions 1521
formatting 123
border style constants 394
ActiveX forms 4
decision pivots 951
OLE objects 1268
scroll boxes 1440
text 634
BorderIcons property
TCustomForm 394
borders
ActiveForms 4
decision pivots 951, 952
edit controls 378
forms 394
grids 425
images 831
list boxes 490, 861
list views 509
OLE objects 1268
panels 566
rebars 266
scroll boxes 1440
speed buttons 1487
static text controls 634
tool windows 1587, 1588
tree views 648
BorderStyle property
TCustomEdit 378
TCustomForm 394
TCustomGrid 425
TCustomListBox 490
TCustomListView 509
TCustomPanel 566
TCustomStaticText 634
TCustomTreeView 648
TDBImage 831
TDBLookupListBox 861
TDecisionPivot 951
TOleContainer 1268
TScrollBox 1440
BorderWidth property
TCustomPanel 566
TDecisionPivot 952
TToolWindow 1587
Borland Assist program 2
Borland Database Engine 70
aliases 682, 1468, 1470, 1471
adding 1464, 1466
changing 1474
testing for 1473
calling 74
configuration modes 1456, 1471
counting records 75
cursor position 729
direct calls 685
drivers 75, 684, 686, 1459, 1465, 1470, 1471, 1472
removing 1468
testing for 686
enabling cached updates 71
event handling 78, 79, 80
isolation levels 690
network control files 1459
processing operations 1461
saving configurations 1475
statement handles 1513
storing temporary tables 1460
table level specification 1545
bounding rectangles
controls 220, 251, 259, 262
list views 510
OCX controls 1290
windowed controls 1672
BoundingRect property
TCustomListView 510
BoundsChanged method
TInplaceEdit 1148
BoundsRect property
TControl 220
boxed edit controls *See* edit controls
BoxRect method
TCustomGrid 438
Break property
TMenuItem 1236
BringToFront method
TApplication 37
TControl 244
Broadcast method
TWinControl 1647
BrowseProperties method
TOleControl 1288
Brush property
TShape 1477
TWinControl 1633
brushes 1633
geometric shapes 1477
bsAutoDetect constant 106
bsBottomLine constant 93
bsBox constant 93
bsDialog constant 394
bsFrame constant 93
bsLeftLine constant 93
bsLowered constant 93
bsNew constant 106
bsNone constant 394, 566, 648, 832, 862, 952, 1268, 1440
list boxes 490
bsRaised constant 93
bsRightLine constant 93
bsSingle constant 394, 566, 648, 832, 862, 952, 1268, 1440
list boxes 490
bsSizeable constant 394
bsSizeToolWin constant 394
bsToolWindow constant 394
bsTopLine constant 93
bsWin31 constant 106
BtnClick method
TDBNavigator 883
buffers 85
data 735, 1061
offset 1052
text 253, 261
Busy property
TReport 1398
button style constants 106
ButtonAutoSize property
TDecisionPivot 952
ButtonCount property
TToolBar 1573
ButtonHeight property
TDecisionPivot 952
TToolBar 1574

buttons 100, 125, 131, 1378, 1485
See also specific type
assigning bitmaps to 102, 103, 105
clicking 108
default 127
grouping 884, 1488
message processing 134
toolbars 1574, 1580
tree views 650
Buttons property
TToolBar 1574
ButtonSpacing property
TDecisionPivot 953
ButtonWidth property
TDecisionPivot 953
TToolBar 1574
bvLowered constant 565
bvNone constant 565
bvRaised constant 565
bytes fields 134, 1615
creating 138, 1618
storing values 136

C

caBottom constant 1014
CacheBlobs property
TBDEDataSet 71
CacheConnections property
TWebApplication 1620
cached updates 1604
applying 83, 85, 693, 1609
canceling 85
clearing buffer 85
datasets 1605
deleted records 1606
enabling 71
inserted records 1606
pending 77, 1051
reconciling 178
restoring previous values 178, 181
storing 1052
CachedUpdates property
TBDEDataSet 71
caDefault constant 1014
caFree constant 404
caHide constant 404
CalAlignment property
TDateTimePicker 770
CalcHintRect method
THintWindow 1129
CalColors property
TDateTimePicker 770
calculated fields
testing for 1038
updating 704, 718, 1052
Calculated property
TField 1038
calendars 770
entering time 773
event handling 773, 774
selecting dates 771
caMinimize constant 404
Cancel buttons 127
Cancel method
TBDEDataSet 85
TDataSet 726
Cancel property
TButton 127
CancelHint method
TApplication 37
canceling data changes 85, 726
CancelLayout method
TCustomDBGrid 333
CancelRange method
TClientDataSet 169
TTable 1549
CancelUpdates method
TBDEDataSet 85
TClientDataSet 169
CanChange method
TCustomListView 524
TCustomTabControl 644
TCustomUpDown 671
CanEdit method
TCustomListView 525
CanEditAcceptKey method
TCustomDBGrid 333
TCustomGrid 438
CanEditModify method
TCustomDBGrid 334
TCustomGrid 438
CanEditShow method
TCustomDBGrid 334
TCustomGrid 439
CanFocus method
TWinControl 1647
CanGridAcceptKey method
TCustomGrid 439
CanModify method
TCustomRadioGroup 582
TDBRadioGroup 891
CanModify property
TBDEDataSet 72
TClientDataSet 152
TDataSet 705
TField 1038
TTable 1540
caNone constant 404
CanPaste property
TOleContainer 1268
canvas objects *See* canvases
Canvas property
TCustomComboBox 282
TCustomControl 299
TCustomForm 395
TCustomListBox 490
TDBCtrlGrid 794
TGraphicControl 1107
THeaderControl 1119
TImage 1136
TStatusBar 1504
canvases 299, 1312
column headers 1119
combo boxes 282
data grids 794
forms 395
graphic controls 1107
image controls 1136
list boxes 490
status bars 1504
Capabilities property
TMediaPlayer 1196
Capacity property
TCustomDataStore 304
Caption property
TControl 221
TDSTableProducer 1014
TMenuItem 1237
CaptionAlignment property
TDSTableProducer 1014
CaptionColor property
TCustomDecisionGrid 359
CaptionFont property
TCustomDecisionGrid 359
captions
bitmap buttons 105
check boxes 276
controls 221, 235, 237, 253
decision grids 359, 363, 364
directory list boxes 989
edit controls 555
list views 529
menu items 1237
panels 564
radio buttons 887, 1380
speed buttons 1489, 1490
table producers 1014, 1023
toolbars 1576
capturing mouse events 231
carriage-return characters 551, 556
Cascade method
TForm 1104

cascading menus *See* menu items; menus
case 379
case-insensitive indexes 167, 1548
CaseSensitive property
TDirectoryListBox 988
caTop constant 1014
cbChecked constant 277
cbGrayed constant 277
cbUnchecked constant 277
cdAnyColor constant 187
cdFullOpen constant 187
cdPreventFullOpen constant 187
CD-ROM drives *See* media players
cdShowHelp constant 187
cdSolidColor constant 187
CellClick method
TCustomDBGrid 334
CellDrawState method
TCustomDecisionGrid 371
CellRect method
TCustomGrid 439
TDrawGrid 1004
Cells property
TCustomDecisionGrid 360
TStringGrid class 1529
CellValueArray method
TCustomDecisionGrid 371
Center property
TAnimate 10
TDBImage 832
TImage 1136
centering images 832
cfmPersistent constant 1457
cfmSession constant 1457
cfmVirtual constant 1457
cgGridLines constant 365
CGI applications 138 to 140, 1619
running 140
cgOutliner constant 365
cgPivotable constant 365
Change Icon OLE dialog box 1277
Change method
TCustomComboBox 291
TCustomEdit 384
TCustomImageList 470
TCustomListView 525
TCustomTabControl 644
TDBEdit 821
TDBMemo 873
TDBRichEdit 898
TDdeServerItem 930
ChangeCount property
TClientDataSet 153
Changed method
TControl 245
Changed tables 53
ChangedCount property
TBatchMove 52
ChangedTableName property
TBatchMove 53
ChangeIconDialog method
TOleContainer 1277
ChangeName method
TComponent 204
ChangeScale method
TControl 245
TCustomForm 411
TScrollingWinControl 1449
TWinControl 1648
changing component names 204
changing data 72, 322, 705, 1038, 1540
canceling changes 85, 726
client datasets 152, 153
data grids 334, 340, 344
testing for changes 710
undoing changes 90, 178, 181
character sets 1524
conversions 59, 114, 380
character strings *See* strings
characters 379
copying and translating 743
CharCase property
TCustomEdit 379
check boxes 141, 145, 275, 777
captions 276
changing current state 280
changing values 779
creating 280, 783
data sources 779
destroying 784
read-only 779
selecting options 147, 277, 780
states 147, 277
CheckBoxes property
TCustomListView 510
CheckBrowseMode method
TDataSet 726
CheckCursor method
TCustomMaskEdit 542
Checked property
TCheckListBox 147
TCustomCheckBox 277
TDateTimePicker 770
TMenuItem 1237
TRadioButton 1380
child components 214
getting 208
child controls 263, 1651, 1661
counting 1634
deleting 1668
getting 417, 421, 1634, 1648, 1649, 1657, 1658, 1671, 1672
message processing 1647, 1662, 1665
moving through 1639, 1640, 1647, 1658, 1660, 1671
painting 1665
scaling 1648, 1670
sorting 1658
child menus *See* menu items; menus
circles 1478
cl3DDkShadow constant 224, 798
cl3DLight constant 224, 799
clActiveBorder constant 223, 798
clActiveCaption constant 223, 798
clAppWorkSpace constant 223, 798
clAqua constant 223, 798
class names
OLE objects 1270
ClassInfo method
TObject 1256
ClassName method
TObject 1256
ClassNameIs method
TObject 1257
ClassParent method
TObject 1257
ClassType method
TObject 1257
clBackground constant 223, 798
clBlack constant 223, 798
clBlue constant 223, 798
clBtnFace constant 223, 798
clBtnHighlight constant 224, 798
clBtnShadow constant 224, 798
clBtnText constant 224, 798
clCaptionText constant 223, 798
CleanupInstance method
TObject 1257
Clear method

TBlobField 117
TCustomComboBox 291
TCustomEdit 385
TCustomImageList 470
TCustomListBox 501
TCustomMaskEdit 542
TCustomRichEdit 601
TField 1060
ClearFields method
TDataSet 727
ClearSelection method
TCustomEdit 385
clFuchsia constant 223, 798
clGray constant 223, 798
clGrayText constant 224, 798
clGreen constant 223, 798
clHighlight constant 223, 798
clHightlightText constant 223, 798
click events
buttons 108, 130
column headers 1121
combo boxes 288
controls 237, 238, 240, 241
overriding 246, 247, 254, 255
data grids 325, 328, 334
database navigator 880, 881
inplace edit controls 1149
list views 519
media players 1208, 1209, 1213, 1217
menus 1243, 1245
speed buttons 1492
up-down controls 668
Click method
TBitBtn 108
TButton 130
TControl 246
TCustomUpDown 671
TMediaPlayer 1213
TMenuItem 1245
TSpeedButton 1492
client applications
connections 583, 917, 1388
data providers 160
client areas
borders 378
coordinate conversions 246, 258
forms 395, 418, 1493
location 221, 251, 1633, 1658
size 221, 222, 1634, 1659
tab controls 638
client datasets 150 to 181
adding records 159, 167, 174
change logs 154, 158, 177
changing 152, 153
constraints 171, 172, 573
counting records 161
creating 170
data providers 572, 1356
deleting indexes 171
destroying 171
event handling 163
filtering 154, 155
getting current record 161, 173
getting data 153, 154, 169, 573
index information 156, 174
adding 167
alternative indexes 157
field counts 156
key size 158
loading 176
locating specific values 173, 176
master/detail relationship 159
read-only 161
record size 161
remote servers 584, 587, 590
saving 179
searching 157, 171
server connections 162
setting current record 175
setting ranges 157, 168, 169, 172, 179, 180
updating 168, 169, 178, 181, 575, 1360
client sockets 181 to 185
asynchronous 182
connecting to 183, 620, 629, 630
creating/destroying 184, 185
disconnecting from 629
host names 621, 631
server addresses 621, 631
standard services 622, 631
ClientDataSets property
TCustomRemoteServer 584
ClientHandle property
TCustomForm 395
ClientHeight property
TControl 221
TCustomForm 395
ClientOrigin property
TControl 221
TWinControl 1633
ClientRect property
TControl 222
TCustomForm 395
TWinControl 1634
ClientToScreen method
TControl 246
ClientType property
TClientSocket 182
ClientWidth property
TControl 222
TCustomForm 395
clInactiveBorder constant 223, 798
clInactiveCaption constant 223, 798
clInactiveCaptionText constant 224, 798
clInfoBk constant 224, 799
clInfoText constant 224, 799
Clipboard
DDE items 930
DDE links 914
Clipboard formats
DDE applications 908, 926
clLime constant 223, 798
clMaroon constant 223, 798
clMenu constant 223, 798
clMenuText constant 223, 798
clNavy constant 223, 798
clock *See* time
clOlive constant 223, 798
CloneCursor method
TClientDataSet 169
Close method
TCustomForm 411
TCustomSocket 629
TDatabase 694
TDataSet 727
TMediaPlayer 1214
TOleContainer 1277
TSession 1466
CloseApplication method
TReport 1403
CloseDatabase method
TSession 1467
CloseDataSets method
TDatabase 694
CloseDialog method
TFindDialog 1085
CloseDimIndexRight method
TDecisionSource 973
CloseIndexFile method
TTable 1549
CloseLink method
TDdeClientConv 912

CloseQuery method
TCustomForm 412
CloseReport method
TReport 1403
CloseUp method
TDBLookupComboBox 851
closing applications 28, 45
closing database sessions 1466, 1467, 1468
closing datasets 727
closing modal forms 398
clPurple constant 223, 798
clRed constant 223, 798
clSilver constant 223, 798
clTeal constant 223, 798
clWhite constant 223, 798
clWindow constant 223, 798
clWindowFrame constant 223, 798
clWindowText constant 223, 798
code 1
Col property
TCustomGrid 426
ColClick method
TCustomListView 525
ColCount property
TCustomDecisionGrid 360
TCustomGrid 426
TDBCtrlGrid 794
ColEnter method
TCustomDBGrid 334
ColExit method
TCustomDBGrid 335
collapsing tree nodes 650, 659
events 653
Collate property
TPrintDialog 1337
collating 1337
Color common dialogs 185 to 189
creating/destroying 188
displaying colors 186
getting selected color 186
opening 188
setting defaults 186
color constants 223
color palettes 252, 833
realizing 256, 419, 1666
Color property
TColorDialog 186
TControl 223
ColoredButtons property
TMediaPlayer 1197
colors 223
See also color constants; color palettes
animation frames 15
containers 232
decision grids 360, 361, 363, 364
grids 428, 797
hint boxes 25
images 463
media player buttons 1197
selecting 185
Cols property
TStringGrid class 1529
column headers 1118 to 1124
See also field names
activating 1120
creating/destroying 1124
drawing surfaces 1119
event handling 1120 to 1122
getting header sections 1120
list views 510, 511, 515
resizing 1121, 1122
selecting header sections 1121
column headings 1042, 1120
data grids 324
Column property
TCustomListView 510
ColumnClick property
TCustomListView 510
ColumnMoved method
TCustomDBGrid 335
TCustomGrid 440
TDrawGrid 1004
columns
data grids *See* data grids
decision grids 360, 362, 372
list views 510, 525, 533
updating 533
menus 1236
radio buttons 579
table producers 1014, 1021, 1024, 1025, 1026
storing 1027
Columns property
TCustomDBGrid 319
TCustomListBox 491
TCustomListView 511
TCustomRadioGroup 579
TDSTableProducer 1014
ColumnsShowing method
TCustomListView 525
ColWidth property
TCustomDecisionGrid 360
ColWidths property
TCustomGrid 426
ColWidthsChanged method
TCustomDBGrid 335
TCustomDecisionGrid 372
TCustomGrid 440
COM interfaces
error handling 1264
getting 199
references 207
combo boxes 189, 281, 784, 845, 1074
See also edit controls; list boxes
adding scroll bars 283
changing items 287, 291
changing values 787
clearing items 291
creating 292, 791
destroying 791
displaying lists 283
drawing surfaces 282
drive lists 1007
entering items 285
event handling 287, 288
getting items 284, 786, 787
message processing 291, 294
painting items 293
read-only 787
selecting items 284, 285, 286
entire blocks 294
size 283
sorting items 286
states 287
styles 286, 294, 787
ComboWndProc method
TCustomComboBox 291
command buttons *See* push buttons; speed buttons
Command property
TMenuItem 1237
commands *See* menu items; menus
Commit method
TDatabase 695
CommitCount property
TBatchMove 53
commits *See* database transactions
CommitUpdates method
TBDEDataSet 85
common dialog boxes 193 to 197
See also dialog boxes
3-D styles 194
accessing 194

closing 195
colors 185
creating/destroying 197
event handling 195, 196, 1083, 1097, 1296, 1297, 1394
finding text 1080
fonts 1093
online help for 195
opening 196, 197
opening files 1291
passwords 1321
printer setup 1341
printing options 1336
replacing text 1393
saving files 1413
saving graphics files 1415
selecting graphics files 1299
templates 195
CommonAVI property
TAnimate 11
ComObject property
TComponent 199
CompareBookmarks method
TBDEDataSet 86
TClientDataSet 170
TDataSet class 728
comparing strings
partial comparisons 74
Component palette 1
ComponentCount property
TComponent 199
ComponentIndex property
TComponent 199
components 1, 198 to 217
See also controls; objects
adding 211
constructing 1
counting 199
creating 1
creating/destroying 204, 206, 207
deleting 213
getting owned 200, 207, 208, 1331
iterating through 199, 213
message processing 207, 212
naming 201, 204, 214, 216
nonvisual 698
objects vs. 1
registering 215
settings *See* properties
states 200, 212, 216
styles 201
updating 215
visual 217
Components property
TComponent 200
ComponentState property
TComponent 200
ComponentStyle property
TComponent 201
ComputerName property
TRemoteServer 1389
ConfigMode property
TSession 1456
configuration files 1475
confirmation messages 878
ConfirmDelete property
TDBNavigator 878
Connect method
TReport 1404
Connected property
TCustomRemoteServer 584
TDatabase 682
TRemoteServer 1389
connections 680
application servers 588
closing 682, 694
DDE applications 906, 917, 921, 927
dropping 590, 686, 1391
event handling 692
maintaining 1459
managing database 1454
opening 682, 687, 696
parameters 687
remote 162, 583, 1388, 1389, 1391, 1392
terminating 1467, 1469
testing 584, 588
Web servers 1620, 1621
Windows sockets 183
active 620, 629, 630
closing 629
opening 630
server types 607, 616, 619
ConnectMode property
TDdeClientConv 907
Constrained property
TQuery 1363
ConstraintErrorMessage property
TField 1039
constraints 573, 1039, 1047
enabling/disabling 87, 171, 172
tables 1540
Constraints property
TCustomProvider 573
TTable 1540
constructing components 1
containers 232, 260, 452
aligning controls 219
colors 232
column headers 1118
help hints 233
OLE objects 1264
radio buttons 578, 884, 1383
testing for 252, 253
text attributes 233
ContainsControl method
TWinControl 1648
Content event
TDataSetTableProducer 747
Content method
TCustomContentProducer 297
TCustomPageProducer 560
TQueryTableProducer 1377
content producers 295 to 298
dispatching messages 295
getting contents 297
message processing 297
ContentFromStream method
TCustomContentProducer 297
TCustomPageProducer 561
ContentFromString method
TCustomContentProducer 297
TCustomPageProducer 561
context menus 234, 1331
alignment 1332
creating/destroying 255, 1335
displaying 1333, 1336
event handling 1334
OLE objects 1268
online help for 1333
returning active object 1333
testing for 252
toolbars 1582
context-sensitive help 40, 1636
dialog boxes 195
menu items 1235, 1240, 1333
OLE applications 1289
ControlAtPos method
TWinControl 1649
ControlCount property
TWinControl 1634
ControlDestroyed method
TApplication 37
controls 217 to 263, 298, 1631
See also components; objects
aligning 219, 1667

boundaries 220, 251, 259, 262
captions 221, 235, 237, 253
changing 245
colors 223, 232
creating/destroying 37, 247, 248, 302
current state 224, 257
disabling/enabling 229, 728, 730, 732
displaying 244, 254, 259, 261, 262
drawing surface 299
event handling 237 to 241, 246, 247, 254, 255
fonts 228, 229, 233
getting active 1419
grouping 562
help hints 230, 233, 235
hiding 236, 262
location 231, 235, 251
origins 221
message processing 236, 245, 247, 255, 256, 263
moving 244, 250
names 232, 260
painting 252, 254, 257
receiving focus 416, 417, 421
scaling 234, 245
setting parent 232, 260
settings *See* properties
size 221, 222, 229, 236
style 225
Controls property
TWinControl 1634
ControlsDisabled method
TDataSet 728
ControlState property
TControl 224
ControlStyle property
TControl 225
ControlType property
TDecisionSource 965
Conv property
TDdeClientConv 907
conversations *See* DDE applications
conversions
ANSI characters 1524
Boolean values 1521
character sets 380
batch moves 59
BLOB fields 114
currency 64
dates 1522
floating-point 64, 763, 1155, 1522
integers 65, 1088, 1522
rich text formats 593, 595, 604
screen coordinates 246, 258
strings 65, 112, 121, 763, 1089, 1156, 1524
variants 66, 97, 112, 122, 764, 1089, 1156
Copies property
TPrintDialog 1337
Copy method
TOleContainer 1277
copying data 743, 1059
copying images 836, 838
copying OLE objects 1268, 1277
copying persistent objects 1329
copying text 385
CopyOnSave property
TOleContainer 1268
CopyParams method
TStoredProc 1516
CopyToClipboard method
TCustomEdit 385
TDBImage 836
TDdeServerItem 930
Count property
TCustomImageList 463
TMenuItem 1238
counters 46
creating 49
counting components 199
counting fields 320, 707, 794
counting forms 1425
counting records 75, 711, 796, 797
client datasets 161
CPS_CANCEL constant 1669
CPS_COMPLETE constant 1669
CPS_CONVERT constant 1669
CPS_REVERT constant 1669
crAppStart constant 227, 1422
crArrow constant 227, 1421
crCross constant 227, 1421
crDefault constant 227, 1421
crDrag constant 227, 1421
Create event
TClientSocket 184
TServerSocket 1454
TStringGrid class 1533
TToolButton 1585
Create method
TActiveForm 7
TAnimate 19
TApplication 38
TAutoIncField 49
TBatchMove 60
TBCDField 69
TBDEDataSet 86
TBevel 95
TBinaryField 99
TBitBtn 109
TBlobField 117
TBooleanField 125
TButton 130
TBytesField 138
TClientDataSet 170
TColorDialog 188
TCommonDialog 197
TComponent 204
TControl 247
TCoolBar 271
TCurrencyField 275
TCustomCheckBox 280
TCustomComboBox 292
TCustomControl 302
TCustomDataStore 313
TCustomDBGrid 335
TCustomDecisionGraph 356
TCustomDecisionGrid 372
TCustomEdit 386
TCustomForm 412
TCustomGrid 440
TCustomGroupBox 456
TCustomHotKey 461
TCustomImageList 470
TCustomLabel 486
TCustomListBox 501
TCustomListView 526
TCustomMaskEdit 543
TCustomMemo 554
TCustomPageProducer 561
TCustomPanel 570
TCustomRadioGroup 582
TCustomRemoteServer 587
TCustomRichEdit 601
TCustomStaticText 636
TCustomTabControl 645
TCustomTreeView 658
TCustomUpDown 671
TCustomWebDispatcher 677
TDatabase 695
TDataModule 701
TDataSet 728
TDataSource 754
TDateField 759
TDateTimeField 768
TDateTimePicker 776
TDBCheckBox 783

TDBComboBox 791
TDBCtrlGrid 802
TDBEdit 822
TDBImage 837
TDBListBox 844
TDBLookupComboBox 851
TDBLookupControl 858
TDBLookupListBox 865
TDBMemo 874
TDBNavigator 883
TDBRadioGroup 891
TDBRichEdit 899
TDBText 905
TDdeClientConv 912
TDdeClientItem 920
TDdeServerConv 924
TDdeServerItem 930
TDecisionCube 934
TDecisionPivot 957
TDecisionSource 974
TDirectoryListBox 993
TDriveComboBox 1012
TDSTableProducer 1022
TField 1060
TFileListBox 1073
TFilterComboBox 1080
TFindDialog 1085
TFloatField 1093
TFontDialog 1098
TGraphicControl 1109
TGraphicField 1113
THeaderControl 1124
THintWindow 1129
TImage 1140
TInplaceEdit 1148
TIntegerField 1160
TMediaPlayer 1214
TMemoField 1230
TMenu 1233
TMenuItem 1245
TNumericField 1254
TObject 1258
TOleContainer 1278
TOleControl 1288
TOpenDialog 1298
TOpenPictureDialog 1302
TPageControl 1307
TPaintBox 1316
TPopupMenu 1335
TProgressBar 1348
TProvider 1360
TQuery 1372
TRadioButton 1383
TReplaceDialog 1396
TReport 1404
TScreen 1429
TScrollBar 1437
TScrollBox 1443
TScrollingWinControl 1449
TSession 1467
TShape 1480
TSmallIntField 1484
TSpeedButton 1492
TSplitter class 1497
TStatusBar 1508
TStoredProc 1517
TStringField 1526
TTable 1550
TTabSheet 1563
TTimeField 1567
TTimer 1570
TToolBar 1579
TToolWindow 1590
TTrackBar 1598
TVarBytesField 1618
TWebApplication 1624
TWinControl 1649
TWordField 1679
CreateColumns method
TCustomDBGrid 336
CreateDataSet method
TClientDataSet 170
CreateEditor method
TCustomDBGrid 336
TCustomGrid 441
CreateForm method
TApplication 38
TWebApplication 1624
CreateHandle method
TApplication 38
TWinControl 1650
CreateLinked method
TDBCtrlPanel 808
CreateLinkToFile method
TOleContainer 1278
CreateNew method
TCustomForm 412
TDataModule 701
CreateObject method
TOleContainer 1278
CreateObjectFromFile method
TOleContainer 1279
CreateObjectFromInfo method
TOleContainer 1279
CreateParams method
TCustomCheckBox 280
TCustomComboBox 292
TCustomEdit 386
TCustomForm 413
TCustomGrid 441
TCustomListBox 502
TCustomListView 526
TCustomMemo 554
TCustomPanel 571
TCustomRichEdit 601
TCustomTabControl 645
TCustomUpDown 672
TInplaceEdit 1149
TWinControl 1650
CreateParented method
TWinControl 1651
CreateSize method
TCustomImageList 471
CreateSubClass method
TWinControl 1651
CreateTable method
TTable 1550
CreateWindowHandle method
TCustomCheckBox 280
TCustomForm 414
TCustomMemo 555
TWinControl 1652
CreateWnd method
TCustomCheckBox 280
TCustomComboBox 292
TCustomDBGrid 336
TCustomEdit 387
TCustomForm 414
TCustomListBox 502
TCustomListView 527
TCustomRichEdit 602
TCustomTabControl 645
TCustomUpDown 672
TScrollingWinControl 1450
TWinControl 1652
creating forms 38
crHandPoint constant 227, 1422
crHelp constant 227, 1422
crHourGlass constant 227, 1421
crHSplit constant 227, 1422
crIBeam constant 227, 1421
crMultiDrag constant 227, 1422
crNo constant 227, 1422
crNoDrop constant 227, 1422
crNone constant 227, 1421
crosstabs 303, 357, 931, 944, 958
adding graphs 348, 937
color options 360, 361, 363, 364
column size 360, 362, 372
counting columns and rows 360, 365
decision sources 964
dimensions 362, 363
displaying values 361, 364

generating subtotals 365
getting information 371
getting values 360, 361, 373, 966, 967
labeling 359, 363, 364
line widths 363
message processing 374
moving items 373, 374
painting 373, 374
row height 362, 365, 375
scrolling 375
selecting cells 375
crSize constant 227, 1421
crSizeNESW constant 227, 1421
crSizeNS constant 227, 1421
crSizeNWSE constant 227, 1421
crSizeWE constant 227, 1421
crSQLWait constant 227, 1422
crUpArrow constant 227, 1421
crVSplit constant 227, 1422
csAcceptsControls constant 226
csAlignmentNeeded constant 225
csAncestor constant 200
csCaptureMouse constant 226
csClicked constant 225
csClickEvents constant 226
csCreating constant 225
csDesigning constant 200
csDesignInteractive constant 226
csDestroying constant 200
csDisplayDragImage constant 226
csDoubleClicks constant 226
csDropDown constant 286, 788
csDropDownList constant 286, 788
csFixedHeight constant 226
csFixedWidth constant 226
csFixups constant 200
csFocusing constant 225
csFramed constant 226
csLButtonDown constant 225
csLoading constant 200
csNoDesignVisible constant 226
csNoStdEvents constant 226
csOpaque constant 226
csOwnerDrawFixed constant 286, 788
csOwnerDrawVariable constant 286, 788
csPaintCopy constant 225
csPalette constant 225
csReading constant 200
csReadingState constant 225
csReplicatable constant 226
csSetCaption constant 226
csSimple constant 286, 788
csUpdating constant 201
csWriting constant 201
Ctl3D property
TCommonDialog 194
TWinControl 1635
currency 61
conversions 64
formatting 66, 1090
currency fields 271
assigning values 1035
creating 275
Currency property
TBCDField 66
TFloatField 1090
current record 797
client datasets 161, 173
setting 175
color options 797
getting 75, 88, 710, 735
moving cursor to 743
CurrentSum property
TDecisionSource 965
CurrentSummary property
TCustomDataStore 305
cursor constants 226, 1420
Cursor property
TControl 226
TScreen 1420
CursorPosChanged method
TDataSet 729
cursors 226, 1420
See also drag cursors; input focus
image lists 473, 474
Cursors property
TScreen 1420
CurValue property
TField 1039
custom components
image lists 462 to 481
custom controls 298 to 303
check boxes 275 to 280
combo boxes 281 to 294
edit boxes 376, 534, 548, 590
grids 317 to 348, 423 to 452
group boxes 452 to 456, 578 to 583
list boxes 488 to 507
list views 507 to 534
panels 562
tabbed notebooks 637 to 646
tree views 646 to 662
up-down controls 663 to 672
CustomColors property
TColorDialog 186
CustomConstraint
TField 1039
customer assistance 2
CustomFormCount property
TScreen 1423
CustomForms property
TScreen 1423
CustomSort method
TCustomListView 527
TCustomTreeView 658
cutting text 387
See also deleting
CutToClipboard method
TCustomEdit 387
TDBImage 837

D

daAbort constant 719, 720, 722
daFail constant 719, 720, 722
daRetry constant 719, 720, 722
data 702
See also data entry; data formats
canceling changes 85, 726
changing 72, 705, 779, 787, 815, 867, 1038, 1540
client datasets 152, 153
data grids 322, 334, 340, 344
testing for 710
copying 743, 1059
deleting 729
displaying 750, 779, 786, 815, 867, 1042
undoing changes 90, 178, 181
data buffers 724, 1061
getting current record 735
offset 1052
data constraints 1039, 1047
data dictionary 1037
data entry 845, 852, 859
grids 341, 443, 444, 1005, 1006
required 1053
restricting 72, 152, 705, 815, 1043
validating 1055, 1062
data filters 73, 709
client datasets 154, 155
options 74, 709
setting text 73

data formats 1037
currency 66, 1090
dates 771
dates/time 764
numeric types 1249, 1251
text 594
data grids 317, 357, 423, 791, 824, 944
See also grids
adding columns 319, 336
background colors 797
changing data 322, 334, 340, 344
context menus 338
counting fields 320, 794
counting records 796, 797
creating/destroying 335, 336, 339, 802
display modes 322
displaying values 342, 348, 795, 803
drawing surface 794
enabling edit mode 795
entering data 341
event handling 325 to 328, 345, 799
getting values 320, 321, 338, 341, 795
highlighting cells 343
in-place editing 334, 336, 342, 347
inserting records 794
keyboard processing 333, 343, 344, 803, 804
labeling columns 324, 1042
links 319, 345
loading 345
locating current record 796, 797
locks 322, 324, 332
decrementing 333, 338, 340, 341
incrementing 332
message processing 346
moving columns and rows 327, 335, 347
orientation 795
painting 320, 327, 328, 337, 339, 340, 796, 799
read-only 323
removing records 793
resetting properties 322
resizing columns 335
resizing records 796, 797
row indicator 321, 337, 346
saving 347
scrolling 346
selecting data 323, 324, 334, 335
setting column attributes 319, 346
updating 332, 341, 347
data integrity 51, 52, 54, 55, 57
data links
decision sources 968
grids 319, 345
remote connections 584, 587, 590
table producers 1015, 1016, 1025
testing 738, 755
data modules 698 to 701, 1423, 1424
creating/destroying 701
event handling 699
Data property
TClientDataSet 153
TCustomProvider 573
TProvider 1357
data providers *See* providers
data sources 749 to 755
batch moves 58
check boxes 779
combo boxes 786
creating/destroying 754
data grids 320, 795
database navigator 878
datasets 153, 705
decision graphs 351
edit controls 818, 823, 870, 895
event handling 752, 753
image controls 832
list boxes 840
lookup controls 856
OLE objects 1272
queries 1364
radio buttons 887
table producers 1015, 1025, 1026
tables 1540, 1544
text controls 903
data states 751
data stores 303 to 317
active dimensions 306, 314
bins 304
caches 304, 306, 307, 308, 315
creating/destroying 313
datasets 305
dimension maps 306, 307, 309, 316
distinct values 313
event handling 309 to 311
formatted strings 314
providing summaries 305, 308, 309, 315, 316
testing dimensions 304
tracking progress 309
variants 314, 316
data types 1041
See also field types
sizes 1054
data-aware controls 777, 815, 829, 852, 867, 884, 901
changing data 779, 787
combo boxes 784, 845
data sources 749
database navigator 876
disabling/enabling 728, 730, 732
displaying data 750, 779, 786
editing data 750, 755
grids 317, 423, 824, 996
list boxes 838
data-aware grids 791
creating 335, 336
display options 322
database aliases 682, 1456, 1470, 1471
adding 1464, 1466
changing 1474
removing 1468
testing for 1473
database components
creating/destroying 695, 696
database connections *See* connections
database drivers 75, 686, 1459, 1465
getting 1470, 1471, 1472
naming 684
removing 1468
testing for 686
database edit controls 815 to 824
checking changes 821
creating/destroying 822
editing 822, 823
getting data 818, 823
message processing 823
multiline 867 to 875
read-only 818
undoing changes 823, 824

database engine *See* Borland Database Engine
database names 683
  returning 1458, 1471
database navigator 876 to 884
  boundaries 884
  creating/destroying 883, 884
  data sources 878
  displaying buttons 878, 879
  displaying messages 878
  event handling 880, 881
  help hints 879
  running 883
database sessions *See* sessions
database transactions 685
  committing 693, 695
  isolation levels 690
  rolling back 697
  starting 697
DatabaseCount property
  TSession 1457
DatabaseName property
  TDatabase 683
databases
  See also data; fields; records; tables
  accessing data sets 684
  adding tables 1538
  connecting to *See* connections
  deleting tables 1551
  emptying tables 1552
  finding active 1458
  finding associated tables 1473
  naming tables 1555
  opening 697
  saving defaults 1475
  status 689
  temporary 689
  transactions *See* database transactions
Databases property
  TSession 1458
DataColor property
  TCustomDecisionGrid 360
DataField property
  TDBCheckBox 779
  TDBComboBox 786
  TDBEdit 818
  TDBImage 832
  TDBListBox 840
  TDBLookupControl 853
  TDBMemo 869
  TDBRadioGroup 886
  TDBRichEdit 894
  TDBText 903
DataFont property
  TCustomDecisionGrid 361
DataLink property
  TCustomDBGrid 319
  TDSTableProducer 1015
DataMode property
  TDdeClientConv 908
DataModuleCount property
  TScreen 1423
DataModules property
  TScreen 1424
DataRequest method
  TCustomProvider 576
Dataset designer 706
DataSet property
  TCustomDataStore 305
  TDataSource 750
  TDSTableProducer 1015
  TField 1040
  TProvider 1357
  TQueryTableProducer 1375
  TUpdateSQL 1605
Dataset property
  TDataSetTableProducer 745
DataSetCount property
  TDatabase 683
datasets 70, 150, 702
  accessing 684
  active 703
  adding records 737
  cached updates 693, 1605
  changing data 72, 152, 153, 705
  closing 694, 727
  counting 683
  creating/destroying 86, 87, 170, 171, 728, 730
  current state 751
  cursor position 729, 743
  data providers 1357
  data sources 153, 705, 750, 755
  data stores 305
  deleting records 729
  displaying records 77
  event handling 713 to 721
  field objects 1040
  filtering 73, 74, 154, 155, 709
  locating specific values 173, 176
  moving through 734, 738, 739, 740, 741
  multi-tiered applications 157
  opening 741
  read-only 161, 1604
  reconciling updates 178
  resynchronizing 742
  searching 157, 171, 710, 732, 733
    filtered records 733, 734
  states 711
  table producers 745, 1015, 1024, 1026, 1375
  updating 91, 168, 169, 181, 575, 742
DataSets property
  TDatabase 684
DataSize property
  TBCDField 67
  TBooleanField 122
  TBytesField 136
  TDateTimeField 764
  TField 1040
  TFloatField 1090
  TIntegerField 1157
  TSmallIntField 1483
  TStringField 1524
DataSource property
  TClientDataSet 153
  TCustomDBGrid 320
  TDataSet 705
  TDBCheckBox 779
  TDBComboBox 786
  TDBCtrlGrid 795
  TDBEdit 818
  TDBImage 832
  TDBListBox 840
  TDBLookupControl 854
  TDBMemo 870
  TDBNavigator 878
  TDBRadioGroup 887
  TDBRichEdit 895
  TDBText 903
  TDSTableProducer 1015
  TQuery 1364
  TTable 1540
DataSumColor property
  TCustomDecisionGrid 361
DataToRawColumn method
  TCustomDBGrid 337
DataType property
  TField 1041
date fields 755 to 759
  creating 759
Date property
  TDateTimePicker 771
date/time fields 760, 1036
  creating 768
  storing 764

DateFormat property
TDateTimePicker 771
DateMode property
TDateTimePicker 771
dates 768
assigning values 762, 763, 764, 766, 1036
conversions 1522
formatting 764, 771
selecting 771
dBASE tables 1546
accessing 1541
aliases 1466
indexes 73, 1542, 1549, 1555
locking 1555, 1557
querying 1364
working directories 684
DblClick method
TControl 247
TInplaceEdit 1149
DDE applications
advise loops 916
clients 906 to 916
Clipboard formats 908, 926
closing connections 912, 924
conversations 906, 917, 921, 927
changing 907
initiating 914
terminating 912
event handling 910, 919, 922, 923, 928
executing portions 908
items 917 to 921, 925
copying 930
exchanging 918, 919, 926, 927, 930
requesting 915
linking 908, 914, 916
automatically 907
poking data 914, 915, 928, 931
running macros 910, 913, 925
servers 909, 921, 925 to 931
specifying topics 909
topics 921
transferring formatting characters 909
DDEConv property
TDdeClientItem 917
DdeFmt property
TDdeClientConv 908
DDEItem property
TDdeClientItem 918
DdeService property
TDdeClientConv 908
DdeTopic property
TDdeClientConv 909
Deactivate method
TCustomForm 415
DeactivateWebModule method
TWebApplication 1624
deallocating *See* freeing
Decision Cube editor 936
decision cubes 931 to 936, 966
activating 932
creating/destroying 934
editing values 936
generating data subsets 935
getting values 958
pivot states 964
summary values 965, 968, 984
as variants 975, 976, 980
choosing 949
decision graphs 348, 937
axes values 967, 968
choosing summary values 949
creating/destroying 356
data sources 351
saving data series 351
decision grids 357, 944
captions 359, 363, 364
choosing summary values 949
color options 360, 361, 363, 364
column size 360, 362, 372
counting columns and rows 360, 365
creating/destroying 372, 373
dimensions 362, 363
displaying values 361, 364
event handling 366, 367, 368
generating subtotals 365
getting information 371
getting values 360, 361, 373, 966, 967
line widths 363
message processing 374
moving items 373, 374
painting 373, 374
row height 362, 365, 375
row/column labels 967, 968
scrolling 375
selecting cells 375
decision pivots 949 to 958
borders 951, 952
creating/destroying 957, 958
dimension buttons 952, 953
getting decision sources 953
layouts 954, 955
resizing 952, 958
decision queries 958 to 963
decision sources 964 to 986
column/row dimensions 966, 967, 968, 979
closing 973
collapsing 974
member count 977
opening 983, 984, 985
placement 978, 983, 985
testing 975, 980
control types 965
creating/destroying 974
dimension states 981
disabling/enabling 973, 983, 984, 985
displaying empty columns/rows 969
distinct values 977
event handling 969, 970, 971
formatted strings 976
getting values 966, 982
labels 977, 979
links 968
message processing 965
pivot states 949 to 958
providing summary values 965, 966, 967, 968, 984
range of cells 978
single values 974
summary names 981
variants 975, 976, 980
DecisionCube property
TDecisionSource 966
DecisionSource property
TCustomDecisionGraph 351
TCustomDecisionGrid 361
TDecisionPivot 953
DECnet connections 619
DefAttributes property
TCustomRichEdit 592
Default property
TButton 127
TMenuItem 1238
DefaultColWidth property
TCustomDecisionGrid 362
TCustomGrid 426
DefaultConverter property
TCustomRichEdit 593

DefaultDrawColumnCell method
TCustomDBGrid 337
DefaultDrawDataCell method
TCustomDBGrid 337
DefaultDrawing property
TCustomDBGrid 320
TCustomGrid 427
DefaultExpression property
TField 1041
DefaultExt property
TOpenDialog 1292
DefaultFields property
TDataSet 706
DefaultHandler method
TControl 247
TCustomDBGrid 338
TCustomForm 415
TObject 1258
TWinControl 1652
DefaultIme property
TScreen 1424
DefaultKbLayout property
TScreen 1424
DefaultRowHeight property
TCustomDecisionGrid 362
TCustomGrid 427
defaults 28
DeferLayout method
TCustomDBGrid 338
DefineFieldMap method
TCustomDBGrid 338
TDSTableProducer 1022
DefineProperties method
TComponent 205
TControl 248
TCustomForm 415
TCustomGrid 442
TCustomImageList 471
TPersistent 1330
DefocusControl method
TCustomForm 416
DefWndProc property
TWinControl 1635
Delete method
TCustomImageList 471
TCustomListView 528
TDataSet 729
TMenuItem 1245
DeleteAlias method
TSession 1468
DeleteColumn method
TCustomGrid 442
DeleteDriver method
TSession 1468
DeleteIndex method
TClientDataSet 171
TTable 1550
DeleteRow method
TCustomGrid 442
DeleteSQL property
TUpdateSQL 1606
DeleteString method
TCustomListBox 503
DeleteTable method
TTable 1551
deleting components 213
deleting data 729, 793
cached updates and 1606
deleting tables 1551
deleting text 385
deleting tree nodes 654
Delta property
TClientDataSet 154
DescriptionsAvailable method
TStoredProc 1517
Deselect method
TInplaceEdit 1149
Designer property
TCustomForm 396
TDataSet 706
DesignInfo property
TComponent 201
DesignOffset property
TDataModule 699
DesignSize property
TDataModule 699
desktop toolbars *See* toolbars
DesktopFont property
TControl 228
Destination property
TBatchMove 53
destination tables 53
Destroy event
TClientSocket 185
TStringGrid class 1533
TToolButton 1586
Destroy method
TApplication 39
TBatchMove 61
TBDEDataSet 87
TBitBtn 109
TCheckListBox 150
TClientDataSet 171
TColorDialog 188
TCommonDialog 197
TComponent 206
TControl 248
TCoolBar 271
TCustomComboBox 293
TCustomControl 302
TCustomDataStore 313
TCustomDBGrid 339
TCustomDecisionGraph 356
TCustomDecisionGrid 373
TCustomForm 416
TCustomGrid 443
TCustomImageList 471
TCustomListBox 503
TCustomListView 528
TCustomMemo 555
TCustomPageProducer 561
TCustomProvider 576
TCustomRadioGroup 582
TCustomRemoteServer 588
TCustomRichEdit 602
TCustomServerSocket 615
TCustomTabControl 645
TCustomTreeView 659
TCustomWebDispatcher 678
TDatabase 696
TDataModule 701
TDataSet 730
TDataSource 754
TDateTimePicker 777
TDBCheckBox 784
TDBComboBox 791
TDBCtrlGrid 802
TDBEdit 822
TDBImage 837
TDBListBox 844
TDBLookupControl 859
TDBMemo 874
TDBNavigator 884
TDBRadioGroup 891
TDBRichEdit 899
TDBText 906
TDdeClientConv 912
TDdeClientItem 921
TDdeServerConv 924
TDdeServerItem 930
TDecisionCube 934
TDecisionPivot 958
TDecisionSource 974
TDirectoryListBox 993
TDriveComboBox 1012
TDSTableProducer 1022
TField 1061
TFileListBox 1073
TFilterComboBox 1080
TFindDialog 1085
TFontDialog 1098
TGraphicControl 1109
THeaderControl 1124
TImage 1140

TMediaPlayer 1214
TMenu 1233
TMenuItem 1246
TObject 1258
TOleContainer 1280
TOleControl 1288
TOpenDialog 1298
TOpenPictureDialog 1302
TPageControl 1308
TPersistent 1330
TPopupMenu 1335
TQuery 1373
TReport 1405
TScreen 1430
TScrollingWinControl 1450
TSession 1468
TShape 1480
TSpeedButton 1492
TStatusBar 1508
TStoredProc 1517
TTable 1551
TTabSheet 1563
TTimer 1571
TToolBar 1579
TWebApplication 1625
TWinControl 1653
DestroyComponents method
TComponent 206
DestroyHandle method
TWinControl 1653
Destroying method
TComponent 206
DestroyObject method
TOleContainer 1280
DestroyWindowHandle method
TCustomForm 416
TWinControl 1653
DestroyWnd method
TCustomComboBox 293
TCustomEdit 387
TCustomListBox 503
TCustomListView 528
TCustomRichEdit 602
TCustomTabControl 645
TWinControl 1654
detaching from databases 682, 694, 1467, 1469
detaching from remote servers 588, 1391
detaching from socket connections 629
detail tables 1544
client datasets 159
device contexts 251, 1659
See also canvases
Device property
TFontDialog 1094
DeviceID property
TMediaPlayer 1197
DeviceType property
TMediaPlayer 1198
dgAlwaysShowEditor constant 322
dgAlwaysShowSelection constant 323
dgCancelOnExit constant 323
dgColLins constant 322
dgColumnResize constant 322
dgConfirmDelete constant 323
dgEditing constant 322
dgIndicator constant 322
dgMultiSelect constant 323
dgRowLines constant 322
dgRowSelect constant 323
dgTabs constant 322
dgTitles constant 322
dialog boxes 23
See also common dialog boxes
displaying 43, 44
message processing 41, 46
multipage 1302, 1558
DialogHandle property
TApplication 23
dialogs *See* dialog boxes
DimensionCount property
TCustomDataStore 306
DimensionMap property
TCustomDataStore 306
DimensionMapCount property
TCustomDataStore 307
Dimensions property
TCustomDecisionGrid 362
directory list boxes 986 to 996
See also list boxes
captions 989
case sensitivity 988, 990, 994
changing directories 995
creating/destroying 993
current directory 988, 989
displaying file names 989
event handling 991
getting path names 994
specifying drives 989
updating 995
directory lists 646, 986, 1067
Directory property
TDatabase 684
TDirectoryListBox 988
TFileListBox 1067
DirLabel property
TDirectoryListBox 989
DirList property
TDriveComboBox 1009
DisableAlign method
TWinControl 1654
DisableAutoRange method
TScrollingWinControl 1450
DisableConstraints method
TBDEDataSet 87
TClientDataSet 171
DisableControls method
TDataSet 730
DisabledImages property
TToolBar 1574
disabling controls 229
disconnecting *See* detaching
disk drive *See* drive lists
Dispatch method
TObject 1259
DispatchAction method
TCustomWebDispatcher 678
DispatchCommand method
TMenu 1234
Dispatcher property
TCustomContentProducer 2 95
dispatcher *See* Web dispatcher
DispatchPopup method
TMenu 1234
Display property
TMediaPlayer 1198
DisplayCase method
TDirectoryListBox 994
DisplayFormat property
TDateTimeField 764
TNumericField 1249
DisplayLabel property
TField 1042
DisplayName property
TField 1042
DisplayRec property
TMediaPlayer 1198
DisplayRect property
TCustomTabControl 638
DisplayText property
TField 1042
DisplayValues property
TBooleanField 123
DisplayWidth property
TField 1043
dmAutomatic constant 229
dmClosed constant 981
dmDrilled constant 981
dmManual constant 229
dmNone constant 981

dmOpen constant 981
dmPaged constant 981
DoActivate method
  TCustomSocket 629
DoAfterDispatch method
  TCustomWebDispatcher 679
DoBeforeDispatch method
  TCustomWebDispatcher 679
DoConnect method
  TCustomRemoteServer 588
  TRemoteServer 1391
DoCreateContent method
  TDSTableProducer 1023
document windows *See* main windows
documentation, printing conventions 2
DoDisconnect method
  TCustomRemoteServer 588
  TRemoteServer 1391
DoEndDrag method
  TControl 248
  TCustomListView 528
DoEnter method
  TWinControl 1654
DoExit method
  TCustomGrid 443
  TWinControl 1655
DoFormatCell method
  TDSTableProducer 1023
DoGetCaption method
  TDSTableProducer 1023
DoHandleException method
  TWebApplication 1625
DoHide method
  TCustomForm 417
DoKey method
  TDBCtrlGrid 803
DoKeyDown method
  TWinControl 1655
DoKeyPress method
  TWinControl 1655
DoKeyUp method
  TWinControl 1656
domain names 621, 631
DoNotify method
  TMediaPlayer 1215
DoObjectVerb method
  TOleControl 1288
DoSetMaxLength method
  TCustomEdit 387
  TCustomRichEdit 603
DoShow method
  TCustomForm 417
DoStartDrag method
  TControl 249
  TCustomListView 528
DoVerb method
  TOleContainer 1280
Down property
  TSpeedButton 1487
  TToolButton 1581
drag and drop operations 250
  canceling 249
  decision grids 373, 374
  event handling 238, 239, 241, 249
  forms 396
  image lists 464, 472, 474, 477, 480
    starting/stopping 470, 473
  initiating 244, 249
  list boxes 504, 506
  list views 511, 528, 530
  modes 228, 260
  states 239, 250
  terminating 248, 250
  tree views 648
drag cursors 228, 252
  hot spots 474
  image lists 464
drag image 472, 480
  getting 530
  hiding 477, 480
DragCanceled method
  TControl 249
  TCustomListBox 504
DragCursor property
  TControl 228
  TCustomImageList 464
DragDrop method
  TControl 249
Dragging method
  TControl 250
Dragging property
  TCustomImageList 464
DragLock method
  TCustomImageList 472
DragMode property
  TControl 228
DragMove method
  TCustomImageList 472
DragOver method
  TControl 250
DragUnlock method
  TCustomImageList 472
draw grids 423, 996
  See also grids
  boundaries 1004
  entering values 1005, 1006
  event handling 999 to 1001
  moving columns and rows 1004, 1005, 1006
  painting 1004
  selecting cells 1005, 1006
Draw method
  TCustomImageList 472
DrawCell method
  TCustomDBGrid 339
  TCustomDecisionGrid 373
  TCustomGrid 443
  TDrawGrid 1004
DrawColumnCell method
  TCustomDBGrid 339
DrawDataCell method
  TCustomDBGrid 340
drawing *See* painting
drawing surfaces *See* canvases
DrawingStyle property
  TCustomImageList 464
DrawItem method
  TCustomComboBox 293
  TCustomListBox 504
DrawOverlay method
  TCustomImageList 473
DrillDimIndex method
  TDecisionSource 974
drive lists 1007 to 1012, 1067
  See also directory list boxes
  case 1009
  creating/destroying 1012
  current drive 1009
  displaying directories 1009
Drive property
  TDirectoryListBox 989
  TDriveComboBox 1009
  TFileListBox 1067
DriverName property
  TDatabase 684
drivers *See* database drivers
drives *See* drive lists
DropConnections method
  TSession 1469
drop-down calendars 770
drop-down lists 293
  See also combo boxes
  adding scroll bars 283
  adding to edit boxes 189
  closing 851
  displaying items 847
  event handling 288
  getting items 284
  opening 283, 847, 851
  size 847

drop-down menus *See* menus
DropDown method
TCustomComboBox 293
TDBLookupComboBox 851
DropDownAlign property
TDBLookupComboBox 847
DropDownCount property
TCustomComboBox 283
DropdownMenu property
TToolButton 1582
DropDownRows property
TDBLookupComboBox 847
DropDownWidth property
TDBLookupComboBox 847
DroppedDown property
TCustomComboBox 283
dropping connections 590, 1391
dropping *See* drag and drop
DropTarget property
TCustomForm 396
TCustomListView 511
TCustomTreeView 648
dsBrowse constant 712, 751
dsCalcFields constant 712, 751
dsCloseAfter constant 367
dsColCaption constant 367
dsColIndicator constant 367
dsColPlus constant 367
dsColValue constant 367
dsData constant 367
dsDragEnter constant 239, 250
dsDragLeave constant 239, 250
dsDragMove constant 239, 250
dsEdit constant 712, 751
dsFilter constant 712, 751
dsFocused constant 464
dsGroupStart constant 367
dsInactive constant 712, 751
dsInsert constant 712, 751
dsNone constant 367
dsNormal constant 464
dsOpenAfter constant 367
dsRowCaption constant 367
dsRowIndicator constant 367
dsRowPlus constant 367
dsRowValue constant 367
dsSelected constant 464
dsSetKey constant 712, 751
dsSum constant 367
dsTransparent constant 464
dsUpdateNew constant 712
dsUpdateOld constant 712
Dynamic Data Exchange *See* DDE

E

ecLowerCase constant 379
ecNormal constant 379
ecUpperCase constant 379
EdgeBorders property
TToolWindow 1587
EdgeInner property
TToolWindow 1588
EdgeOuter property
TToolWindow 1588
edit boxes *See* edit controls
edit controls 376 to 390, 815, 1027
adding drop-down lists 189
clearing 385, 387
copying text 385
creating/destroying 386, 387
data sources 818, 823, 870
event handling 382
getting selected text 388
masked 534 to 548, 1187
multiline 548 to 557, 867, 1222
password protecting 380
pasting text 389
read-only 381, 597, 818, 870, 895
rich text 590 to 606, 891, 1408
selecting text 378, 379, 381, 385, 389, 390
size 378, 379, 387
updating 380, 382, 384
Edit method
TCustomListView 529
TDataSet 731
TDataSource 755
edit mode 450, 795
keyboard processing 823, 874
EditButtonClick method
TCustomDBGrid 340
EditCanModify method
TCustomMaskEdit 543
TDBEdit 822
TInplaceEdit 1150
EditChange method
TPasswordDialog 1326
EditFormat property
TNumericField 1251
EditHandle property
TCustomComboBox 283
editing ActiveX controls 1355
editing data 731, 750, 755, 815, 1055
data grids 333, 334, 336, 340, 342, 347
grids 427, 430, 438, 439, 441, 444, 446, 450
editing text 376, 822
masked edit controls 542, 543, 544
memo fields 867 to 875
rich text controls 891
tree views 654, 661
EditKey method
TClientDataSet 171
TTable 1551
EditMask property
TCustomMaskEdit 536
TField 1043
EditMaskPtr property
TField 1045
EditMode property
TDBCtrlGrid 795
Editor property
TDSTableProducer 1016
editor window *See* inplace edit controls
EditorMode property
TCustomGrid 427
EditRangeEnd method
TClientDataSet 172
TTable 1551
EditRangeStart method
TClientDataSet 172
TTable 1552
EditText property
TCustomMaskEdit 538
Eject method
TMediaPlayer 1215
ellipses 1478
embedding OLE objects 1264
emptying tables 1552
EmptyTable method
TTable 1552
EnableAlign method
TWinControl 1657
EnableAutoRange method
TScrollingWinControl 1450
EnableConstraints method
TBDEDataSet 87
TClientDataSet 172
EnableControls method
TDataSet 732
Enabled property
TControl 229
TDataSource 750
TMenuItem 1239
TTimer 1568

EnabledButtons property
TMediaPlayer 1199
enabling controls 229
encryption *See* passwords
EndDrag method
TControl 250
TCustomImageList 473
EndLayout method
TCustomDBGrid 340
TDSTableProducer 1024
EndPage property
TReport 1398
EndPos property
TMediaPlayer 1200
EndUpdate method
TCustomDBGrid 341
TDSTableProducer 1024
enhanced metafiles *See* metafiles
EnumCtlProps method
TPropertyPage 1354
EOF property
TDataSet 706
Error method
TCustomSocket 629
Error property
TMediaPlayer 1200
ErrorMessage property
TMediaPlayer 1200
errors
See also exceptions
data providers 1357
masked edit controls 548
media players 1200
OLE applications 1264
socket connections 615, 625, 629
esLowered constant 1588
esNone constant 1588
esRaised constant 1588
Event method
TCustomSocket 629
event timer 1567 to 1571
creating/destroying 1570, 1571
enabling/disabling 1568
events 1
animation controls 16, 17
applications 29 to 34
bitmap buttons 108
calendars 773, 774
capturing mouse 231
client datasets 163
column headers 1120 to 1122
combo boxes 287, 288
common dialogs 195, 196, 1083, 1097, 1296, 1297, 1394
custom sockets 623 to 627, 629
data grids 325 to 328, 345, 799
data modules 699
data providers 574, 576, 1357
data sources 752, 753
data stores 309 to 311
database engine 78, 79, 80
database navigator 880, 881
database sessions 1462
databases 692
datasets 713 to 721
DDE applications 910, 919, 922, 923, 928
decision grids 366, 367, 368
decision sources 969, 970, 971
directory lists 991
draw grids 999 to 1001
edit controls 382
field objects 1056 to 1057
file lists 1070
filter combo boxes 1077
forms 403 to 408
graphics controls 1314
image controls 1138
image lists 466, 470
list boxes 148
list views 518 to 520
lookup combo boxes 848, 849
media players 1208, 1209
menu objects 1243, 1334
nonwindowed controls 237 to 241
overriding 246, 247, 254, 255
OLE objects 1273, 1274
page producers 1310
panels 567
radio buttons 889
rebars 268, 269
remote servers 585, 586
rich text edit controls 597 to 598
screen objects 1428
scroll bars 1434, 1435
scroll boxes 1441
server sockets 608 to 613, 615, 616, 617, 618
splitters 1495, 1497, 1498
status bars 1505, 1506
tab controls 641
table producers 1018, 1019
toolbars 1577
track bars 1596
tree views 652 to 655
up-down controls 668
Web dispatcher 675, 676
windowed controls 1641 to 1643
overriding 1654, 1655, 1656, 1662, 1663
exceptions 30
applications 39, 45
BLOB fields 119
Web applications 1625
Exclusive property
TTable 1541
ExecProc method
TStoredProc 1518
ExecSQL method
TQuery 1373
TUpdateSQL 1610
executable files 23
running 45
Execute method
TBatchMove 61
TColorDialog 188
TCommonDialog 197
TFindDialog 1086
TFontDialog 1098
TOpenDialog 1299
TOpenPictureDialog 1302
TPrintDialog 1341
TPrinterSetupDialog 1343
TSaveDialog 1415
TSavePictureDialog 1417
ExecuteMacro method
TDdeClientConv 913
TDdeServerConv 925
ExecuteMacroLines method
TDdeClientConv 913
executing batch moves 60, 61
ExeName property
TApplication 23
expanding tree nodes 649, 650, 659
events 654, 655
ExpIndex property
TBDEDataSet 73
Explorer-style dialog boxes 1299
expression indexes 73, 1548
extended ASCII characters 59, 114, 1524
ExtendedSelect property

TCustomListBox 492

F

fdAnsiOnly constant 1095
fdApplyButton constant 1095
fdBoth constant 1094
fdEffects constant 1096
fdFixedPitchOnly constant 1096
fdForceFontExist constant 1096
fdLimitSize constant 1096
fdNoFaceSel constant 1096
fdNoOEMFonts constant 1096
fdNoSimulations constant 1096
fdNoSizeSel constant 1096
fdNoStyleSel constant 1096
fdNoVectorFonts constant 1096
fdPrinter constant 1094
fdScalableOnly constant 1096
fdScreen constant 1094
fdShowHelp constant 1096
fdTrueTypeOnly constant 1096
fdWysiwyg constant 1096
FetchAll method
  TBDEDataSet 88
fetching records 88
FetchOnDemand property
  TClientDataSet 154
field definitions 707
field names 1046
  See also column headers
  displaying 1042
  getting 732, 736, 1052
field objects 1032 to 1065
  accessing 707
  assigning values 1053, 1060
  changing data 1038
  clearing 727
  copying values 1059
  counting 707
  creating/destroying 1060, 1061
  current values 1039, 1055
  default 706
  default values 1041
  determining type 1045
  displaying data 1042, 1043
  event handling 1056 to 1057
  finding 733
  getting data 1040, 1050, 1051, 1064
  getting specific 736, 779, 787
  hiding 1056
  moving through 1061
  null values 1048, 1060
  read-only 1053
  reordering 1046, 1048
  storing 1040
  updating 1051, 1052
  validating 1055, 1062, 1064
Field property
  TDBCheckBox 779
  TDBComboBox 787
  TDBEdit 818
  TDBImage 833
  TDBListBox 841
  TDBLookupControl 854
  TDBMemo 870
  TDBRadioGroup 887
  TDBRichEdit 895
  TDBText 903
field types 1060, 1064
  binary 95, 109, 134, 1615
  Boolean 119
  currency 61, 271
  date/time 755, 760, 1563
  integrity violations 52, 57
  numeric 46, 1086, 1153, 1247, 1481, 1675
  string 1519
  text 1226
field values 708
  getting 90, 739
  setting 743
FieldAddress method
  TObject 1260
FieldByName method
  TDataSet 732
FieldCount property
  TCustomDBGrid 320
  TDataSet 707
FieldDefs property
  TDataSet 707
FieldKind property
  TField 1045
FieldName property
  TField 1046
FieldNo property
  TField 1046
Fields property
  TCustomDBGrid 321
  TDataSet 707
fields *See* field objects; specific field types
FieldValues property
  TDataSet 708
file attributes 1068
file filters 1074, 1293, 1294
  See also file masks
file list boxes 1065
  adding images 1069
  changing items 1070
  creating/destroying 1073
  directory structures 1067
  displaying items 1067, 1068
  drives 1067
  event handling 1070
  filtering 1069, 1074
  selecting items 1068
  updating 1072, 1073
file lists 646
file masks 1069, 1076, 1077
  See also file filters
file names
  animation controls 11
  list boxes 989, 1068
file types 1068
FileCompareText method
  TDirectoryListBox 994
FileEdit property
  TFileListBox 1067
FileEditStyle property
  TOpenDialog 1292
FileList property
  TDirectoryListBox 989
  TFilterComboBox 1076
FileLoad method
  TCustomImageList 473
FileName property
  TAnimate 11
  TFileListBox 1068
  TMediaPlayer 1201
  TOpenDialog 1292
files 1068
  See also file list boxes
Files property
  TOpenDialog 1293
FileType property
  TFileListBox 1068
filter combo boxes 1074 to 1080
  changing items 1077
  creating/destroying 1080
  displaying items 1076
  event handling 1077
  file masks 1076, 1077
Filter property
  TBDEDataSet 73
  TClientDataSet 154
  TDataSet 709
  TFilterComboBox 1076
  TOpenDialog 1293
Filtered property
  TBDEDataSet 73
  TClientDataSet 155
  TDataSet 709
FilterIndex property

TOpenDialog 1294
FIlterOptions property
TBDEDataSet 74
TClientDataSet 155
TDataSet 709
filters 73, 709, 1074
See also file filters; masks
client datasets 154, 155
options 74
setting text 73
Find common dialogs 1080 to 1086
closing 1085
creating/destroying 1085
position 1081, 1083
running 1086
search strings 1081
setting defaults 1082
FindCaption method
TCustomListView 529
FindComponent method
TComponent 207
FindData method
TCustomListView 529
FindDatabase method
TSession 1469
FindField method
TDataSet 733
FindFirst method
TDataSet 733
finding *See* searching
FindItem method
TMenu 1234
FindKey method
TClientDataSet 173
TTable 1552
FindLast method
TDataSet 733
FindNearest method
TClientDataSet 173
TTable 1553
FindNext method
TDataSet 734
FindNextControl method
TWinControl 1657
FindNextPage method
TPageControl 1308
FindPrior method
TDataSet 734
FindText method
TCustomRichEdit 603
FindText property
TFindDialog 1081
First method
TDataSet 734
FixedColor property
TCustomGrid 428
FixedCols property
TCustomDecisionGrid 363
TCustomGrid 428
FixedOrder property
TCoolBar 267
FixedRows property
TCustomDecisionGrid 363
TCustomGrid 428
FixedSize property
TCoolBar 267
FixupTabList method
TWinControl 1658
fkCalculated constant 1046
fkCommand constant 1234
fkData constant 1046
fkHandle constant 1235
fkInternalCalc constant 1046
fkLookup constant 1046
fkShortCut constant 1235
Flat property
TDBNavigator 878
TSpeedButton 1487
TToolBar 1575
flat-file datasets 150
floating-point fields 1086
creating 1093
storing values 1090
floating-point numbers 1036, 1088
assigning values 1088, 1089, 1090, 1091
conversions 64, 763, 1155, 1522
formatting 1090
precision 1091
FlushSchemaCache method
TDatabase 696
Fmt property
TDdeServerItem 926
FnDataCols property
TDecisionSource 966
focus *See* input focus
FocusControl method
TCustomForm 417
TField 1061
FocusControl property
TCustomLabel 483
TCustomStaticText 634
Focused method
TWinControl 1658
Font common dialogs 1093
devices 1094
running 1098
setting defaults 1095
setting font size 1095
Font property
TControl 229
TFontDialog 1094
fonts 228, 229, 1425
controls 233
decision grids 361
icons 1426
rich text edit controls 592, 595
Fonts property
TScreen 1425
Footer property
TDSTableProducer 1016
form controls 390, 1099 to 1105
activating/deactivating 392, 393, 403, 405, 411, 415
borders 394
cascading 1104
changing active form 411
client areas 395, 418, 1493
closing 398, 403, 404, 411, 412
counting 1423
creating 38, 405, 412, 413, 414
currently opened 1425
destroying 416
displaying 401, 402, 408, 417, 422
drawing surface 395
event handling 403 to 408
getting active 1419, 1420, 1423
getting children 417, 421
hiding 406, 417, 418
icons 394, 418, 1104
keyboard processing 397
loading 418
main menus 398
message processing 415, 419, 423
painting 407, 419
position 399
printing 400, 420
properties 415, 420
receiving focus 416, 417, 421, 1104, 1105
resizing 407, 420, 1493
saving 230
scaling 399, 401, 411
setting parents 421
style 396
tiling 1105
Form designer 201, 396
design-mode flags 214

form style constants 396
FormatChars property
TDdeClientConv 909
formatted memo fields 1226
See also memo fields; memos
formatting data 1037
formatting dates 764, 771
formatting numbers 1249, 1251
as currency 66, 1090
formatting text 594
formatting time 764
FormCount property
TScreen 1425
Forms property
TScreen 1425
forms *See* form controls; MDI forms
FormStyle property
TCustomForm 396
Found property
TDataSet 710
FrameCount property
TAnimate 11
FrameHeight property
TAnimate 12
Frames property
TMediaPlayer 1201
FrameWidth property
TAnimate 12
frDisableMatchCase constant 1082
frDisableUpDown constant 1082
frDisableWholeWord constant 1082
frDown constant 1082
Free method
TObject 1260
FreeBookmark method
TDataSet 735
freeing memory 420, 1260
FreeInstance method
TObject 1260
FreeNotification method
TComponent 207
FreeOnRelease method
TComponent 207
Frequency property
TTrackBar 1592
frFindNext constant 1082
frHideMatchCase constant 1082
frHideUpDown constant 1082
frHideWholeWord constant 1082
frMatchCase constant 1082
FromPage property
TPrintDialog 1337
frReplace constant 1082
frReplaceAll constant 1082
frShowHelp constant 1082
frWholeWord constant 1082
fsMDIChild constant 396
fsMDIForm constant 396
fsNormal constant 396
fsStayOnTop constant 396
ftArchive constant 1068
ftAutoInc constant 1041, 1054, 1062
ftBCD constant 1041, 1054, 1063
ftBlob constant 113, 118, 1041, 1054, 1063
ftBoolean constant 1041, 1054, 1062
ftBytes constant 1041, 1054, 1063
ftCurrency constant 1041, 1054, 1063
ftCursor constant 1041, 1054, 1063
ftDate constant 1041, 1054, 1063
ftDateTime constant 1041, 1054, 1063
ftDBaseOle constant 113, 119, 1041, 1054, 1063
ftDirectory constant 1068
ftFloat constant 1041, 1054, 1063
ftFmtMemo constant 113, 118, 1041, 1054, 1063
ftGraphic constant 113, 118, 1041, 1054, 1063
ftHidden constant 1068
ftInteger constant 1041, 1054, 1063
ftMemo constant 113, 118, 1041, 1054, 1063
ftNormal constant 1068
ftParadoxOle constant 113, 118, 1041, 1054, 1063
ftReadOnly constant 1068
ftSmallInt constant 1054, 1062
ftSmallint constant 1041, 1054
ftString constant 1041, 1054, 1063
ftSystem constant 1068
ftTime constant 1041, 1054, 1063
ftTypedBinary constant 113, 119, 1041, 1054, 1063
ftUnknown constant 1041, 1054, 1063
ftUnkown constant 1041, 1054
ftVarBytes constant 1041, 1054, 1063
ftVolumeID constant 1068
ftWord constant 1041, 1054, 1062
FullCollapse method
TCustomTreeView 659
FullExpand method
TCustomTreeView 659
FullRepaint property
TCustomPanel 567

G

geometric shapes 1476
Get_Data method
TCustomProvider 576
TProvider 1360
Get2DDataAsVariant method
TDecisionSource 975
GetActiveDim method
TDecisionSource 975
GetAliasDriverName method
TSession 1470
GetAliasNames method
TSession 1470
GetAliasParams method
TSession 1470
GetBitmap method
TCustomImageList 474
GetBookmark method
TDataSet 735
GetCells method
TCustomDecisionGrid 373
GetChildOwner method
TComponent 208
GetChildParent method
TComponent 208
GetChildren method
TComponent 208
TCustomForm 417
TCustomRadioGroup 582
TWinControl 1658
GetClientOrigin method
TControl 251
TWinControl 1658
GetClientRect method
TControl 251
TCustomForm 418
TWinControl 1659
GetColField method
TCustomDBGrid 341
GetConfigParams method
TSession 1471
GetConnected method
TCustomRemoteServer 588

TRemoteServer 1392
GetCurrentRecord method
TBDEDataSet 88
TClientDataSet 173
TDataSet 735
GetData method
TField 1061
GetDataAsString method
TDecisionSource 976
GetDataAsVariant method
TDecisionSource 976
GetDatabaseNames method
TSession 1471
GetDataSet method
TDSTableProducer 1024
GetDataSource method
TDSTableProducer 1025
GetDetailSQL method
TDecisionCube 935
GetDeviceContext method
TControl 251
TWinControl 1659
GetDimensionMemberCount method
TCustomDataStore 313
TDecisionSource 977
GetDimensionName method
TCustomDataStore 314
TDecisionSource 977
GetDragImages method
TControl 252
TCustomListView 530
GetDriverNames method
TSession 1471
GetDriverParams method
TSession 1472
GetEditLimit method
TCustomDBGrid 341
TCustomGrid 443
GetEditMask method
TCustomDBGrid 341
TCustomGrid 444
TDrawGrid 1005
GetEditText method
TCustomDBGrid 342
TCustomGrid 444
TDrawGrid 1005
GetEnumPropDesc method
TOleControl 1288
GetExtensionVersion property
TISAPIApplication 1162
GetFieldList method
TDataSet 736
GetFieldNames method
TDataSet 736
GetFieldValue method
TCustomDBGrid 342
GetFirstEditChar method
TCustomMaskEdit 543
GetFormImage method
TCustomForm 418
GetGetSocketEvent method
TCustomServerSocket 615
GetGetThreadEvent method
TCustomServerSocket 615
GetGridHeight method
TCustomGrid 444
GetGridWidth method
TCustomGrid 445
GetGroup method
TDecisionSource 978
GetGroupExtent method
TDecisionSource 978
GetHelpContext method
TMenu 1235
TOleControl 1289
GetHitTestInfoAt method
TCustomTreeView 660
GetHotSpot method
TCustomImageList 474
GetIcon method
TCustomImageList 474
GetIconMetaPict method
TOleContainer 1280
GetIDsOfNames method
TComponent 209
GetImageBitmap method
TCustomImageList 475
GetImages method
TCustomImageList 475
GetIndex method
TDecisionSource 979
GetIndexInfo method
TBDEDataSet 89
TClientDataSet 174
GetIndexNames method
TClientDataSet 174
TTable 1553
GetInstRes method
TCustomImageList 474
GetInterface method
TObject 1260
GetInterfaceEntry method
TObject 1261
GetInterfaceTable method
TObject 1261
GetItemAt method
TCustomListView 530
GetItemData method
TCustomListBox 504
GetItemIndex method
TCustomListView 530
GetItemPath method
TDirectoryListBox 994
GetLabelText method
TCustomLabel 487
GetLastEditChar method
TCustomMaskEdit 543
GetMaskBitmap method
TCustomImageList 476
GetMaxChars method
TCustomMaskEdit 544
GetMemberAsString method
TCustomDataStore 314
TDecisionSource 979
GetMemberAsVariant method
TCustomDataStore 314
TDecisionSource 980
GetMemoryUsage method
TCustomDataStore 315
GetMetaData method
TCustomProvider 576
GetNamePath method
TComponent 209
TPersistent 1331
GetNearestItem method
TCustomListView 530
GetNextEditChar method
TCustomMaskEdit 544
GetNextItem method
TCustomListView 531
GetNextPacket method
TClientDataSet 174
GetNodeAt method
TCustomTreeView 661
GetObjectVerbs method
TOleControl 1289
GetOle2AcceleratorTable method
TMainMenu 1185
GetOnClientError method
TCustomServerSocket 615
GetOnClientEvent method
TCustomServerSocket 616
GetOnThreadEnd method
TCustomServerSocket 616
GetOnThreadStart method
TCustomServerSocket 616
GetOwner method
TComponent 209
TPersistent 1331
GetPalette method
TControl 252
GetParentComponent method
TComponent 209

TControl 252
GetParentHandle method
TWinControl 1659
GetPassword method
TSession 1472
GetPopupMenu method
TControl 252
GetPriorEditChar method
TCustomMaskEdit 544
GetPropDisplayString method
TOleControl 1289
GetPropDisplayStrings method
TOleControl 1290
GetProvider method
TCustomProvider 577
TCustomRemoteServer 589
TRemoteServer 1392
GetProviderNames method
TCustomRemoteServer 589
TRemoteServer 1392
GetRecords method
TCustomProvider 577
TProvider 1360
GetResource method
TCustomImageList 476
GetResults method
TStoredProc 1518
GetRowState method
TDecisionSource 980
GetSearchString method
TCustomListView 532
GetSel method
TCustomMaskEdit 545
GetSelLength method
TCustomEdit 388
TCustomRichEdit 603
GetSelStart method
TCustomEdit 388
TCustomRichEdit 603
GetSelText method
TCustomEdit 388
TCustomRichEdit 604
GetSelTextBuf method
TCustomEdit 388
TCustomRichEdit 604
GetServerType method
TCustomServerSocket 616
GetSQL method
TDecisionCube 935
GetState method
TDecisionSource 981
GetStaticRect method
TOpenDialog 1299
GetStoredProcNames method
TSession 1472
GetSummaryAsString method
TCustomDataStore 315
GetSummaryAsVariant method
TCustomDataStore 316
GetSummaryName method
TCustomDataStore 316
TDecisionSource 981
GetTableNames method
TSession 1473
GetTabOrderList method
TDBCtrlGrid 803
TWinControl 1660
GetTextBuf method
TControl 253
GetTextLen method
TControl 253
TCustomMaskEdit 545
GetThreadCacheSize method
TCustomServerSocket 617
GetTopParentHandle method
TWinControl 1660
GetTypeInfo method
TComponent 210
GetTypeInfoCount method
TComponent 210
GetValue method
TDecisionSource 982
GetValueArray method
TDecisionSource 982
GetValueIndex method
TDecisionSource 982
gkAppend constant 803
gkCancel constant 803
gkDelete constant 803
gkDown constant 803
gkEditMode constant 803
gkEnd constant 803
gkHome constant 803
gkInsert constant 803
gkLeft constant 803
gkNextTab constant 803
gkNull constant 803
gkPageDown constant 803
gkPageUp constant 803
gkPriorTab constant 803
gkRight constant 803
gkScrollDown constant 803
gkScrollUp constant 803
gkUp constant 803
globally unique identifiers 1260
Glyph property
TBitBtn 102
TSpeedButton 1487
glyphs 102, 1487
See also graphic controls; images
goAlwaysShowEditor constant 431
goColMoving constant 431
goColSizing constant 431
goDrawFocusSelected constant 431
goEditing constant 431
goFixedHorzLine constant 431
goFixedVertLine constant 431
goHorzLine constant 431
goRangeSelect constant 431
goRowMoving constant 431
goRowSelect constant 431
goRowSizing constant 431
goTabs constant 431
goThumbTracking constant 431
GotoBookmark method
TDataSet 736
GotoCurrent method
TClientDataSet 175
TTable 1554
GotoKey method
TClientDataSet 175
TTable 1554
GotoNearest method
TClientDataSet 175
TTable 1554
goVertLine constant 431
graphic controls 1105 to 1109, 1312, 1476
canvases 1107
creating/destroying 1109
painting 1109, 1316
graphics fields 1110 to 1113
creating 1113
graphics files 1299, 1415
graphics objects 1316
graphs 348, 937
axes values 967, 968
choosing summary values 949
creating/destroying 356
data sources 351
decision sources 964
saving data series 351
Grid property
TInplaceEdit 1145
GridHeight property
TCustomGrid 429
GridLineColor property
TCustomDecisionGrid 363
GridLines property
TCustomListView 511

GridLineWidth property
TCustomDecisionGrid 363
TCustomGrid 429
grids 357, 423, 944, 1526
See also data grids; draw grids; string grids
adding data 443, 444
backgrounds 428
borders 425
closing 443
column widths 426, 440
counting columns/rows 426, 432, 434
creating/destroying 440, 441, 443
deleting columns and rows 442
display options 431
editing 427, 430, 438, 439, 441, 444, 446, 450
getting current column 426
getting specific cells 447
getting top row 434
getting visible columns 430, 434
in-place editing 1145
input masks 444
invalidating regions 445, 446
keyboard processing 438, 439, 446
line widths 429
moving columns and rows 440, 448, 449
moving through 433
painting 427, 443, 448
row size 427, 432, 449
scrolling 428, 433, 449, 451, 452
selecting cells 432, 433, 450
setting properties 431, 442
size 429, 444, 445, 451
adjusting 438
updating 450, 452
GridWidth property
TCustomGrid 429
group boxes 452, 1113
creating/destroying 456, 582, 891
radio buttons 578, 884, 1383
Grouped property
TToolButton 1582
GroupIndex property
TMenuItem 1239
TSpeedButton 1488
grouping controls 562
GroupLayout property
TDecisionPivot 954
Groups property
TDecisionPivot 954
GroupSpacing property
TDecisionPivot 955
GUIDs 1260

H

Handle property
TApplication 24
TBDEDataSet 74
TCommonDialog 194
TCustomImageList 465
TDatabase 685
TMainMenu 1183
TMenu 1231
TMenuItem 1240
TSession 1458
TWinControl 1636
HandleAllocated method
TCustomImageList 477
TWinControl 1660
HandleException method
TApplication 39
HandleMessage method
TApplication 39
HandleNeeded method
TCustomImageList 477
TWinControl 1661
HandleRequest method
TWebApplication 1625
handles 38, 414
animation controls 13
check boxes 280
database engine 74, 685, 1368, 1458
DDE applications 907
dialog boxes 194
drop-down lists 284
edit boxes 283
hint window 1130
image lists 465, 466, 477
main forms 24
MDI clients 395
menu objects 1183, 1231, 1232, 1240
multiline edit controls 555
non-VCL windows 1638
parent windows 1659, 1660
reports 1400
stored procedures 1513
windowed controls 1636, 1640, 1660, 1674
creating 1650, 1652, 1661
destroying 1653, 1668
HandleTag method
TCustomPageProducer 562
HasConstraints property
TField 1047
HasParent method
TComponent 210
TControl 253
hcAlt constant 459
hcCtrl constant 459
hcCtrlAlt constant 459
hcNone constant 459
hcShift constant 459
hcShiftAlt constant 459
hcShiftCtrl constant 459
hcShiftCtrlAlt constant 459
Header property
TDSTableProducer 1016
headings *See* column headings
Height property
TControl 229
TCustomImageList 465
TScreen 1426
help files 24, 397
accessing 40
Help Hints 24
activating hint window 1128, 1129
color options 25
controls 230, 233, 235
database navigator 879
displaying 26, 37, 41, 1125
enabling/disabling 27
event handling 31, 34
hiding 25, 41
hiding hint window 1129
menu items 1241
updating window 29
window size 1129
Help systems 24, 40, 397
context sensitivity 40, 1636
dialog boxes 195
event handling 30, 406
menu items 1235, 1240, 1333
OLE applications 1289
providing jumps 40
help topics 40
HelpCommand method
TApplication 40
HelpContext method
TApplication 40
HelpContext property
TCommonDialog 195
TMenuItem 1240
TPopupMenu 1333

TWinControl 1636
HelpFile property
TApplication 24
TCustomForm 397
HelpJump method
TApplication 40
hidden files 1068
Hide method
TControl 254
TCustomForm 418
TInplaceEdit 1150
HideDragImage method
TCustomImageList 477
HideEditor method
TCustomGrid 445
HideHint method
TApplication 41
HideScrollBars property
TCustomRichEdit 593
HideSelection property
TCustomEdit 379
TCustomListView 512
TCustomRichEdit 593
TCustomTreeView 648
hierarchical lists 646
hierarchical menus *See* menus
hierarchy 1262
HighlightCell method
TCustomDBGrid 343
highlighting cells 343
Hint property
TApplication 24
TControl 230
TMenuItem 1241
HintColor property
TApplication 25
HintHidePause property
TApplication 25
HintMouseMessage method
TApplication 41
HintPause property
TApplication 26
hints 24
color options 25
controls 230, 233, 235
database navigator 879
displaying 26, 37, 41, 1125
enabling/disabling 27
event handling 31, 34
hiding 25, 41
menu items 1241
updating window 29
Hints property
TDBNavigator 879
HintShortPause property
TApplication 26
HistoryList property
TOpenDialog 1294
HitTest property
TCustomGrid 430
hkAlt constant 459
hkCtrl constant 459
hkExt constant 459
hkShift constant 459
HookMainWindow method
TApplication 41
horizontal scroll bars 1432, 1445
grids 433
horizontal track bars 1594
HorzScrollBar property
TScrollingWinControl 1445
host applications 3
Host property
TCustomSocket 621
hosts 621, 631
hot keys 456 to 462, 1130, 1242
See also accelerators
creating 461
invalid modifiers 458
key combinations 458
testing for 1235
valid modifiers 459
hot spots 474
HotImages property
TToolBar 1575
HotKey property
TCustomHotKey 458
HotTrack property
TCustomListView 512
TCustomTabControl 638
THeaderControl 1120
htAbove constant 660
htBelow constant 660
HTML commands 1023
table producers 1013
HTML templates 557
converting 560, 561, 1309
creating/destroying 561
storing 558
transparent tags 558, 562, 1310
HTMLDoc property
TCustomPageProducer 558
HTMLFile property
TCustomPageProducer 558
htNowhere constant 660
htOnButton constant 660
htOnIcon constant 660
htOnIndent constant 660
htOnItem constant 660
htOnLabel constant 660
htOnRight constant 660
htOnStateIcon constant 660
htToLeft constant 660
htToRight constant 660
HTTP request messages 674, 678, 1162, 1620
caching 1620, 1621
dispatching 673, 1626
processing 1621, 1623, 1624, 1625
querying 1374 to 1378
HTTP response messages 675
content producers 295 to 298
dispatching 679
table producers 744
HttpExtensionProc property
TISAPIApplication 1162
Hypertext Markup Language
See HTML

I

Icon property
TApplication 26
IconFont property
TScreen 1426
Iconic property
TOleContainer 1269
IconOptions property
TCustomListView 512
icons
fonts 1426
forms 394, 418
image lists 469, 473, 474, 478, 479
list views 512, 513, 515, 516, 524
MDI forms 1104
minimized applications 26, 1622
OLE objects as 1269, 1277, 1280
IDABORT constant 43
IDCANCEL constant 43
IDIGNORE constant 43
idle applications 32
IDNO constant 43
IDOK constant 43
IDRETRY constant 43
IDYES constant 43
IIDs 1260
image controls 829, 1134 to 1140
See also images
associating with data 832, 833

autosizing 1135
borders 831
centering images 832, 1136
copying 836
creating/destroying 837, 1140
deleting contents 837
displaying contents 833
automatically 831
drawing surfaces 1136
event handling 1138
loading images 838, 1136
painting 833
pasting 838
read-only 833
resizing 834, 1135, 1137
specifying images 1137
transparency 1137
image lists 462, 1141 to 1144
adding items 465, 468, 469, 477, 478
associating with trees 648, 651
changing 466, 470, 479, 481
copying 469
counting items 463
creating/destroying 466, 470, 471, 477
deleting items 470, 471
dragging items 464, 472, 474, 477, 480
starting/stopping 470, 473
event handling 466, 470
getting items 474, 475, 476
loading resources 473, 474, 476, 480
moving items 470, 473, 478
painting 463, 464, 465, 469, 472, 473
replacing items 479
size 463, 465, 466
ImageIndex property
TToolButton 1582
images 462, 1134
See also image controls
collections 1141
colors 463
drag cursors 228
file lists 1069
list boxes 490
list views 512, 513, 515, 516, 524
mouse cursor 226, 1420, 1461
mouse pointer 252
rebars 266, 267
speed buttons 1487, 1489
layout constants 1489
styles 464
toolbars 1574, 1575, 1576, 1582
Images property
TCoolBar 267
TCustomTreeView 648
TToolBar 1575
ImageType property
TCustomImageList 465
imAlpha constant 1637
imChinese constant 1637
imClose constant 1637
imDisable constant 1637
imDontCare constant 1637
IME editor 1424, 1426, 1636, 1637
activating 1673
composition window 1669, 1673
resetting 1669
ImeMode property
TWinControl 1636
ImeName property
TWinControl 1637
Imes property
TScreen 1426
imHanguel constant 1637
imHira constant 1637
imKata constant 1637
imOpen constant 1637
ImportedConstraint property
TField 1047
imSAlpha constant 1637
imSHanguel constant 1637
imSKata constant 1637
InactiveCount property
TWebApplication 1621
Increment property
TCustomUpDown 666
incremental searches 856
IncrementalDisplay property
TImage 1136
increments 46
Indent property
TCustomTreeView 649
TToolBar 1576
indenting text 551
See also aligning
Indeterminate property
TToolButton 1583
index files
closing 1549
opening 1555
Index property
TField 1048
IndexDefs property
TClientDataSet 156
TTable 1541
indexes 1541
arrays *See* arrays
client datasets 156, 174
adding 167
alternative 157
deleting 171
field counts 156
key size 158
defining 1542, 1543
deleting 1550
getting keys 1541
key size 75
partial keys 1544
primary keys 46
retrieving information 89
returning names 1553
setting ranges 1543
testing keys 1048
type constants 1548
IndexFieldCount property
TClientDataSet 156
TTable 1541
IndexFieldNames property
TClientDataSet 156
TTable 1542
IndexFields property
TClientDataSet 156
TTable 1542
IndexFiles property
TTable 1542
IndexName property
TClientDataSet 157
TTable 1543
IndexOf method
TMenuItem 1246
IndicatorOffset property
TCustomDBGrid 321
InheritsFrom method
TObject 1262
InitialDir property
TOpenDialog 1294
Initialize method
TApplication 42
TCustomImageList 477
TWebApplication 1626
initializing
button controls 109
check boxes 280
combo boxes 292

edit controls 386
graphic controls 1109
multiline edit controls 554, 874
nonwindowed controls 247
objects 1258, 1262
rich text edit controls 601
tree views 658
Web applications 1626
windowed controls 1649
InitialValues property
TReport 1398
initiating actions 100
InitInstance method
TObject 1262
inplace activation 1267
inplace edit controls 430, 439, 446, 1144
creating 441, 1148, 1149
data grids 334, 336, 342, 347
displaying 450, 452, 1150
enabling 1150, 1152
hiding 445
keyboard processing 1150, 1151
message processing 1153
moving 1148, 1151, 1152
redrawing 1150
resizing 1148
selecting items 1149
updating contents 1152
InplaceEditor property
TCustomGrid 430
input focus
data grids 797, 803
event handling 1641, 1654, 1655
form objects 1104, 1105
forms 416, 417, 421
inplace edit controls 1152
list views 512
moving 1639, 1640
setting 1672
static text controls 634
testing for 1647, 1658
input masks 341, 1005
grids 444
input method editor 1424, 1426, 1636, 1637
activating 1673
composition window 1669, 1673
resetting 1669
input parameters *See* stored procedures
Insert method
See also Add method
TCustomImageList 477
TDataSet 737
TMenuItem 1246
Insert Object OLE dialog box 1281
InsertComponent method
TComponent 211
InsertControl method
TWinControl 1661
InsertIcon method
TCustomImageList 478
insertion point *See* cursors
InsertMasked method
TCustomImageList 478
InsertObjectDialog method
TOleContainer 1281
InsertRecord method
TDataSet 737
InsertSQL property
TUpdateSQL 1606
InstanceSize method
TObject 1262
instantiating objects 1256, 1257, 1263
integer fields 1153, 1481, 1675
creating 1160, 1484, 1679
storing values 1157, 1483
integers 61, 271, 1036, 1156
assigning values 1155, 1156, 1157, 1158
conversions 65, 1088, 1522
IntegralHeight property
TCustomListBox 492
integrity violations 51, 54, 55
data types 52, 57
interface identifiers *See* IIDs
interfaces 217
getting 1260
returning entries 1261
InternalDataSource property
TDSTableProducer 1016
interrupts 44
Interval property
TTimer 1568
InTransaction property
TDatabase 685
Invalidate method
TControl 254
TInplaceEdit 1150
TWinControl 1661
InvalidateCell method
TCustomGrid 445
InvalidateCol method
TCustomGrid 445
InvalidateEditor method
TCustomGrid 446
InvalidateRow method
TCustomGrid 446
InvalidKeys property
TCustomHotKey 458
Invoke method
TComponent 211
IP addresses 621, 631
IProvider interface 160, 573, 1392
creating 577
getting 589
IPX/SPX connections 619
IsAlias method
TSession 1473
ISAPI applications 1160 to 1163, 1619
closing 1163
handling response messages 1162
names 1162
version numbers 1162
IsBlob method
TBlobField 117
TField 1062
IsControl property
TControl 230
IsControlMouseMsg method
TWinControl 1662
IsCustomProperty method
TOleControl 1290
isCut constant 531
isDropHilited constant 531
IsEditing method
TCustomListView 532
TCustomTreeView 661
IsEmpty method
TDataSet class 738
isFocused constant 531
IsHintMsg method
THintWindow 1129
IsLinkedTo method
TDataSet 738
TDataSource 755
IsMasked property
TCustomMaskEdit 538
isNone constant 531
IsNull property
TBlobField 113
TField 1048
isolation levels 690
See also database transactions
isSelected constant 531

IsShortCut method
TMenu 1235
IsSQLBased property
TDatabase 686
IStorage interface 1273
isUnique constant 1548
IsValidChar method
TField 1062
ItemAtPos method
TCustomListBox 505
ItemFocused property
TCustomListView 512
ItemHeight property
TCustomComboBox 283
TCustomListBox 492
ItemIndex property
TCustomComboBox 284
TCustomListBox 493
TCustomRadioGroup 579
ItemRect method
TCustomListBox 505
Items property
TCustomComboBox 284
TCustomListBox 493
TCustomListView 513
TCustomRadioGroup 580
TCustomTreeView 649
TDBComboBox 787
TDBListBox 841
TDBRadioGroup class 887
TMenu 1231
TMenuItem 1241
itImage constant 465
itMask constant 465
IUnknown interface 217
ixCaseInsensitive constant 167, 1548
ixDescending constant 167, 1548
ixExpression constant 1548
ixPrimary constant 1548

J

jumps 40
See also Help systems
justification *See* alignment

K

KeepConnection property
TDatabase 686
KeepConnections property
TSession 1459
key fields 854, 855, 1048, 1541
client datasets 158
exclusive keys 1543
primary values 46
searching on 1544
testing 1048
key violations 51, 54, 55
keyboard constants 803
keyboard events 1642, 1643
data grids 333, 343, 344, 803, 804
database edit controls 823, 899, 900
forms 397
grids 438, 439, 446
inplace edit controls 1150, 1151
masked edit controls 545, 546
multiline edit controls 556, 874
overriding 1655, 1656, 1662, 1663
up-down controls 665
keyboard layouts 1424
keyboard shortcuts *See* accelerators
KeyDown method
TCustomDBGrid 343
TCustomGrid 446
TCustomMaskEdit 545
TDBCtrlGrid 804
TDBEdit 823
TDBMemo 874
TDBRichEdit 899
TInplaceEdit 1150
TWinControl 1662
KeyExclusive property
TClientDataSet 157
TTable 1543
KeyField property
TDBLookupControl 854
KeyFieldCount property
TClientDataSet 157
TTable 1544
KeyFields property
TField 1048
KeyPress method
TCustomDBGrid 344
TCustomGrid 446
TCustomMaskEdit 545
TCustomMemo 556
TDBEdit 823
TDBMemo 874
TDBRichEdit 900
TInplaceEdit 1151
TWinControl 1663
KeyPreview property
TCustomForm 397
KeySize property
TBDEDataSet 75
TClientDataSet 158
KeyUp method
TCustomMaskEdit 546
TInplaceEdit 1151
TWinControl 1663
KeyValue property
TDBLookupControl 855
Keyviol tables 54, 55
writing to 51
KeyViolCount property
TBatchMove 54
KeyViolTableName property
TBatchMove 55
Kind property
TBitBtn 103
TDateTimePicker 771
TScrollBar 1432

L

LabelColor property
TCustomDecisionGrid 364
LabelFont property
TCustomDecisionGrid 364
labels 481 to 488, 1164 to 1167
See also captions; column headers; field names
aligning text 482, 483
creating 486
crosstabs 359, 363, 364
drawing 487
message processing 487
resizing 483
selecting 483
setting accelerators 484
size 487
transparent 484
wordwrapping 485
LabelSumColor property
TCustomDecisionGrid 364
language drivers 75, 686, 1459, 1465
getting 1470, 1471, 1472
removing 1468
LargeChange property
TScrollBar 1432
LargeImages property
TCustomListView 513
Last method
TDataSet 738
LaunchType property
TReport 1399

Layout property
TBitBtn 104
TCustomLabel 483
TSpeedButton 1489
LayoutChanged method
TCustomDBGrid 344
TDSTableProducer 1025
LayoutLock property
TCustomDBGrid 322
lbOwnerDrawFixed constant 495
lbOwnerDrawVariable constant 495
lbStandard constant 495
Left property
TControl 231
TFindDialog 1081
LeftCol property
TCustomGrid 430
Length property
TMediaPlayer 1201
libraries 1
Lines property
TCustomMemo 550
TCustomRichEdit 594
TDdeClientItem 918
TDdeServerItem 926
LineSize property
TTrackBar 1593
LinkActive method
TCustomDBGrid 345
TDSTableProducer 1025
Linked property
TOleContainer 1269
linking OLE objects 1264
links
data grids 319, 345
DDE applications 907, 908, 912, 914, 916
decision sources 968
remote connections 584, 587, 590
table producers 1015, 1016, 1025
testing 738, 755
list boxes 145, 488, 768, 838, 859, 1167
See also combo boxes
adding items 493, 841
assigning values 506
borders 490, 861
checking items 147
creating 501, 502, 844
deleting strings 503
destroying 150, 503, 844
directory structures 986
displaying items 492, 496, 498, 504, 505
drawing surfaces 490
event handling 148
file lists 1065
getting items 504
getting values 840, 841
multi-column 491
read-only 841, 862
removing items 501, 503, 506
selecting items 492, 493, 494, 505
sorting items 494
tab widths 495
list items *See* list views
List property
TToolBar 1576
list views 507, 1172
activating items 512
adding items 509, 520
borders 509
boundaries 510
changing items 518, 525, 529, 532
checking items 510
column headers 510, 511, 515
columns 510, 525, 533
updating 533
creating/destroying 526, 527, 528
deleting items 519, 528
displaying items 511, 517, 532
drop targets 511
event handling 518 to 520
getting items 513, 516, 529, 530, 531
hiding items 512
images 512, 513, 515, 516, 524
iterating through 530
matching items 532
message processing 532, 534
moving items 511, 528, 530
origins 516
position constants 530, 531
properties 524
read-only 513
scrolling 532
selecting entire row 514
selecting items 512, 513, 514, 518, 650
sorting items 515, 519, 523, 527
states 516, 531
styles 517
updating 533
listeners 606, 625, 1452
getting 615, 616
setting 617, 618
ListField property
TDBLookupControl 855
ListFieldIndex property
TDBLookupControl 856
ListHandle property
TCustomComboBox 284
lists
See also list views
hierarchical 646
scrollable 145, 189, 281, 488, 1167
ListSource property
TDBLookupControl 856
ListVisible property
TDBLookupComboBox 847
live result sets 1366
Loaded method
TComponent 211
TCustomDBGrid 345
TCustomForm 418
TCustomMemo 556
TCustomRemoteServer 589
TCustomSocket 630
TMediaPlayer 1215
LoadFromFile method
TBlobField 117
TClientDataSet 176
TCustomTreeView 661
TOleContainer 1281
LoadFromStream method
TBlobField 117
TClientDataSet 176
TCustomTreeView 662
TOleContainer 1281
loading resources
images 473, 474, 480, 1136
pictures 838
LoadMemo method
TDBMemo 875
TDBRichEdit 900
LoadPicture method
TDBImage 838
Local property
TQuery 1364
Locale property
TBDEDataSet 75
TDatabase 686
TSession 1459
locales 75, 686, 1459
Locate method

TBDEDataSet 89
TClientDataSet 176
TDataSet 739
Locked property
TCustomPanel 567
locks
data grids 322, 324, 332, 333, 338, 340, 341
table 1555, 1557
table producers 1021
LockTable method
TTable 1555
LogChanges property
TClientDataSet 158
logical fields *See* Boolean fields
logical values *See* Boolean values
Login dialog box 687, 1177
LoginPrompt property
TDatabase 687
long strings 1257
lookup combo boxes 845 to 852
See also combo boxes
creating 851
displaying values 847, 848
event handling 848, 849
opening/closing lists 847, 851
testing status 847
lookup controls 852
creating/destroying 858, 859
getting values 853, 854, 856
message processing 859
read-only 856
lookup fields 1049, 1051
getting data 1050, 1051
multi-tiered applications 177
refreshing 1063
lookup list boxes 859 to 865
adding items 862
borders 861
creating 865
read-only 862
selecting items 862
Lookup method
TBDEDataSet 90
TClientDataSet 177
TDataSet 739
Lookup property
TField 1049
lookup tables *See* lookup combo boxes; lookup controls
lookup values 90, 739
LookupCache property
TField 1049
LookupDataSet property
TField 1050
LookupKeyFields property
TField 1050
LookupList property
TField 1051
LookupResultField property
TField 1051
lowercase characters 379
lrDefaultColor constant 475, 476
lrDefaultSize constant 475, 476
lrFromFile constant 475, 476
lrMap3DColors constant 475, 476
lrMonoChrome constant 475, 476
lrNone constant 475, 476
lrTransparent constant 475, 476
ltDefault constant 1399
ltDesignTime constant 1399
ltRunTime constant 1399

M

macros (DDE) 910, 913, 925
main menus 398, 1182
OLE applications 399, 1185, 1186
main windows 26
bringing to front 37
MainForm property
TApplication 26
MainWndProc method
TWinControl 1664
Mappings property
TBatchMove 55
Margin property
TBitBtn 104
TSpeedButton 1489
Mask property
TFileListBox 1069
TFilterComboBox 1077
masked edit controls 534 to 548, 1187
clearing 542
creating 543
displaying text 538
editing 542, 543, 544
error handling 548
keyboard processing 545, 546
resetting 546
selecting text 545, 547
size 539, 544, 545
state 538
testing 538
valid text 536, 539, 546, 547
Masked property
TCustomImageList 465
masks 536, 1043, 1045
See also filters; masked edit controls
file filters 1069, 1076, 1077
grids 341, 444, 1005
image lists 465, 468, 469, 475, 476, 477, 478, 479
MaskState property
TCustomMaskEdit 538
master tables 1544
client datasets 159
data sources 1544
MasterFields property
TClientDataSet 159
TTable 1544
MasterSource property
TClientDataSet 159
TTable 1544
Max property
TCustomUpDown 666
TProgressBar 1345
TScrollBar 1433
TTrackBar 1593
MaxCells property
TCustomDataStore 307
MaxConnections property
TWebApplication 1621
MaxDate property
TDateTimePicker 772
MaxDimensions property
TCustomDataStore 308
MaxFontSize property
TFontDialog 1095
maximizing applications 33, 44
MaxLength property
TCustomComboBox 285
TCustomEdit 379
TCustomMaskEdit 539
MaxPage property
TPrintDialog 1337
MaxRecords property
TReport 1399
MaxRows property
TDSTableProducer 1017
MaxSummaries property
TCustomDataStore 308
MaxValue property
TBCDField 67
TFloatField 1090
TIntegerField 1157
mbBarBreak constant 1237

mbBreak constant 1237
mbNone constant 1237
MCI drivers *See* media players
MCI error codes 1200
MDI forms
accessing 397
active window 393
arranging icons 1104
cascading 1104
counting children 397
menus 401
tiling 401, 1105
MDIChildCount property
TCustomForm 397
MDIChildren property
TCustomForm 397
MeasureItem method
TCustomComboBox 294
TCustomListBox 505
Media Control Interface *See* MCI
media files 1201, 1220
media players 1191 to 1222
advancing 1201, 1216, 1221
capabilities 1196
closing 1214
creating/destroying 1214
current position 1203
device ID/type 1197, 1198
displaying onscreen 1217, 1221
displaying output 1198
ejecting loaded medium 1215
enabling/disabling buttons 1194, 1199, 1212
error handling 1200
event handling 1208, 1209
message processing 1202, 1203, 1215, 1216
modes 1201
pausing 1217, 1218, 1219
playable tracks 1206
recording 1220
resetting position 1219
rewinding 1196, 1201, 1213, 1220
running 1201, 1207, 1217, 1218
automatically 1195
starting position 1204, 1206
stopping 1200, 1221
time formats 1204
track lengths 1206
visible buttons 1207
memo fields 1226 to 1230
See also memos; multi-line edit controls
confirming changes 873
creating 1230
displaying data 869, 875
editing 867 to 875
getting data 869, 870
undoing changes 874
memory
data stores 304, 307, 308, 315
freeing 420, 1260
memos 548 to 557, 894, 1222 to 1226
See also memo fields; multi-line edit controls
scrolling 550, 557
menu bars 1182
menu items 1231, 1235
adding text 1237
checking 1237
columns 1236
context numbers 1240
counting 1238
creating/destroying 1245, 1246
enabling/disabling 1239
event handling 1243
getting 1234
hiding 1243
ID numbers 1237
inserting 1244, 1246
parent menus 1241
position 1246
removing 1245, 1247
selecting 1234, 1242
menu objects 1182, 1230, 1235, 1331
event handling 1243, 1334
Menu property
TCustomForm 398
MenuIndex property
TMenuItem 1241
menus 1182, 1230 to 1235
See also menu items; menu objects
accelerators 1235, 1242
accessing submenus 1231, 1241
creating/destroying 1233
dispatch processing 1234
forms 398
MDI forms 401
merging 1183, 1185, 1186, 1239
OLE applications 399
online help for 1235
Merge method
TMainMenu 1185
MergeChangeLog method
TClientDataSet 177
message boxes 42
displaying 43, 44
message handlers 1258, 1259
MessageBox method
TApplication 42
messages 42
See also error messages
applications 32, 39, 44
buttons 134
combo boxes 291, 294
data grids 346
database navigator 878
decision grids 374
decision sources 965
dialog boxes 41, 46
edit controls 823
forms 415, 419, 423
inplace edit controls 1153
labels 487
list views 532, 534
lookup controls 859
media players 1202, 1203, 1215, 1216
multiline edit controls 875, 900
nonwindowed controls 236, 245, 247, 255, 256, 263
overriding 1675
processing 1258, 1259
producers 297
table producers 1026
up-down controls 672
Web dispatcher 680
windowed controls 1635, 1647, 1652, 1662, 1664, 1665, 1675
metadata 576
sending in packets 159
metafiles 1280
MethodAddress method
TObject 1263
MethodName method
TObject 1263
methods 1263
MIDI sequencer *See* media players
Min property
TCustomUpDown 666
TProgressBar 1345
TScrollBar 1433

TTrackBar 1593
MinDate property
TDateTimePicker 772
MinFontSize property
TFontDialog 1095
Minimize method
TApplication 43
minimized applications 26, 43
displaying titles 28
event handling 33
Web servers 1622
MinPage property
TPrintDialog 1338
MinSize property
TSplitter class 1494
MinValue property
TBCDField 67
TFloatField 1091
TIntegerField 1158
MMNotify method
TMediaPlayer 1216
mnemonic *See* accelerators
modal controls 258
modal dialog boxes *See* modal forms
modal forms 421
closing 127, 398
displaying 422
ModalResult property
TButton 127
TCustomForm 398
Mode property
TBatchMove 56
TMediaPlayer 1201
modeless dialog boxes 23
modes 1485
Modified method
TPropertyPage 1355
Modified property
TBlobField 114
TCustomEdit 380
TDataSet 710
TOleContainer 1269
Modifiers property
TCustomHotKey 459
ModifyAlias method
TSession 1474
ModifySQL property
TUpdateSQL 1607
money fields *See* currency fields
monikers *See* OLE objects
mouse
cursor constants 226, 1420
images 252, 1420, 1461
message processing 1662
moving 240, 255
mouse events 240, 241
See also click events
capturing 231
custom grids 430, 447, 451
data grids 325, 327, 328, 345
decision grids 373, 374
Help Hints 31
list views 512
overriding 254, 255
splitters 1497
MouseCapture property
TControl 231
MouseCoord method
TCustomGrid 447
MouseDown method
TControl 254
TCustomDBGrid 345
TCustomDecisionGrid 373
TCustomGrid 447
TSplitter class 1497
MouseMove method
TControl 255
TCustomDecisionGrid 374
TCustomGrid 447
TSplitter class 1497
MouseToCell method
TDrawGrid 1005
MouseUp method
TControl 255
TCustomDBGrid 345
TCustomDecisionGrid 374
TCustomGrid 447
TSplitter class 1497
Move method
TCustomImageList 478
TInplaceEdit 1151
MoveBy method
TDataSet 739
MoveColRow method
TCustomGrid 448
MoveColumn method
TCustomGrid 448
MovedCount property
TBatchMove 57
MoveDimIndexes method
TDecisionSource 983
MoveRow method
TCustomGrid 448
moving mouse 240, 255
moving scroll bars 1432, 1434
mpCanEject constant 1196
mpCanPlay constant 1196
mpCanRecord constant 1196
mpCanStep constant 1196
mpNotReady constant 1202
mpOpen constant 1202
mpPaused constant 1202
mpPlaying constant 1202
mpRecording constant 1202
mpSeeking constant 1202
mpStopped constant 1202
mpUsesWindows constant 1196
mrAbort constant 398
mrAll constant 398
mrCancel constant 398
mrIgnore constant 398
mrNo constant 398
mrNone constant 398
mrOk constant 398
mrRetry constant 398
mrYes constant 398
msDBSetText constant 538
msMasked constant 538
msReEnter constant 538
multi-column list boxes 491
multiline edit controls 548 to 557, 891, 1222
aligning text 549, 556
creating 554, 555, 874
displaying items 869, 875
editing 867 to 875
getting items 550, 556, 869, 870
indenting text 551
keyboard processing 556, 874
loading 556
message processing 875, 900
naming 555
new lines 551, 556
read-only 870, 895
scrolling 550, 557
updating 873
wordwrapping 551, 557
MultiLine property
TCustomTabControl 639
multimedia devices 1191
See also media players
multipage dialogs 1302, 1558
accessing pages 1305
adding pages 1307
changing pages 1308, 1309
counting pages 1304
current page 1304
multiple document interface *See* MDI
MultiSelect property
TCustomListBox 493
TCustomListView 513

multi-tiered applications 150
adding records 159, 167, 174
counting records 161
data providers 572, 1356
enabling/disabling constraints 171, 172
event handling 163
filtering records 154, 155
getting current record 161, 173
getting data 153, 154, 169, 573, 1357
at runtime 170
specific records 1360
indexes 156, 174
adding 167
alternative indexes 157
deleting 171
field counts 156
key size 158
loading client datasets 176
locating specific values 173, 176
logging changes 154, 158, 177
lookup fields 177
master/detail tables 159
read-only datasets 161
record size 161
remote connections 162, 583, 1388, 1389, 1391, 1392
saving 179
searching 157, 171
setting current record 175
setting ranges 157, 168, 169, 172, 179, 180
updating 168, 169, 178, 181

N

Name property
TComponent 201
TControl 232
TOleControl 1285
names
See also captions
components 201, 204, 214, 216
controls 232, 260
navigator *See* database navigator
nColDims property
TDecisionSource 966
nDataRows property
TDecisionSource 967
nDims property
TDecisionSource 967
NetFileDir property
TSession 1459
network control files 1459
NewInserted property
TOleContainer 1270
NewInstance method
TObject 1263
NewValue property
TField 1051
Next method
TDataSet 740
TMediaPlayer 1216
non-blocking connections 607
non-production indexes 1542, 1549
opening 1555
nonvisual components 698
nOpenColDims property
TDecisionSource 967
nOpenRowDims property
TDecisionSource 967
NormalizeTopMosts method
TApplication 43
notebook dividers *See* tab sets
notebook pages *See* page sets
notification messages *See* Notification method
Notification method
TComponent 212
TControl 255
TCustomContentProducer 2 97
TCustomDBGrid 346
TCustomDecisionGrid 374
TCustomForm 419
TCustomLabel 487
TCustomListView 532
TCustomUpDown 672
TCustomWebDispatcher 680
TDBEdit 823
TDBLookupControl 859
TDBMemo 875
TDBRichEdit 900
TDSTableProducer 1026
TMediaPlayer 1216
Notify property
TMediaPlayer 1202
NotifyControls method
TWinControl 1665
NotifyValue property
TMediaPlayer 1203
nRowDims property
TDecisionSource 968
NSAPI applications 1160
nSums property
TDecisionSource 968
null values 1060
testing for 113, 1048
numbers *See* floating-point numbers; integers; numeric fields
numeric fields 1247
creating 1254
currency 61, 271
floating-point 1086
formatting 1249, 1251
incremental 46
integers 1153, 1481, 1675
NumGlyphs property
TBitBtn 105
TSpeedButton 1489
nvAborted constant 1203
nvFailure constant 1203
nvSuccessful constant 1203
nvSuperseded constant 1203

O

Object Inspector 1, 1331
getting text strings 1331
Object Linking and Embedding *See* OLE
object pointers 1256, 1257
object types 1256, 1257
getting 1256
hierarchy 1262
testing for 1257
ObjectMenuItem property
TCustomForm 399
object-oriented programming 1
ObjectPropertiesDialog method
TOleContainer 1282
objects 1, 1255 to 1264, 1327
See also components; controls
components vs. 1
creating/destroying 1258, 1260, 1263
OLE *See* OLE objects
publishing 1260
Objects property
TStringGrid class 1530
ObjectVerbs property
TOleContainer 1270
OCX controls *See* OLE custom controls
ODBC socket 686
odChecked constant 288
odDisabled constant 288, 497
odFocused constant 288, 497
odGrayed constant 288

odSelected constant 288, 497
OEM characters 380
OEMConvert property
TCustomEdit 380
ofAllowMultiSelect
constant 1295
ofCreatePrompt constant 1295
ofExtensionDifferent
constant 1295
ofFileMustExist constant 1295
Offset property
TField 1052
ofHideReadOnly constant 1295
ofNoChangeDir constant 1295
ofNoDereferenceLinks
constant 1295
ofNoLongNames constant 1295
ofNoNetworkButton
constant 1295
ofNoReadOnlyReturn
constant 1295
ofNoTestFileCreate
constant 1295
ofNoValidate constant 1295
ofOldStyleDialog constant 1295
ofOverwritePrompt
constant 1295
ofPathMustExist constant 1295
ofReadOnly constant 1296
ofShareAware constant 1296
ofShowHelp constant 1296
OKButtonClick method
TPasswordDialog 1327
OldStreamFormat property
TOleContainer 1270
OldValue property
TField 1052
OLE applications
error handling 1264
main menus 399, 1185, 1186
online help 1289
running 1282
toolbars 567
OLE automation 42
OLE components 1284
OLE containers 1264 to 1284
OLE controls 393
OLE custom controls
(OCX) 1284
associations 1285
boundaries 1290
creating/destroying 1288
display IDs 1289, 1290, 1291
getting verbs 1289
initiating actions 1288
names 1285
online help for 1289
properties 1288, 1290
showing information
about 1291
OLE documents 1267
OLE interfaces 1271
display strings 1289, 1290,
1291
error handling 1264
low-level accesses 1273
OLE objects
accessing 1273
activating 1267
borders 1268
changing 1269
context menus 1268
copying 1268, 1277
creating 1278, 1279, 1281
data sources 1272
destroying 1280
displaying 1269, 1277
embedding 1264
event handling 1273, 1274
formats 1270
getting class names 1270
getting verbs 1270, 1271,
1284
hosts 1271
inplace activation 399
linking 1264
loading/unloading 1277,
1281
pasting 1268, 1282
saving 1283
sending verbs 1280
setting properties 1282
size 1271
states 1272
temporary storage 1268
testing status 1269, 1270
updating 1283
OleClassName property
TOleContainer 1270
OleFormObject property
TForm 399
OleObject property
TOleContainer 1271
TOleControl 1285
TPropertyPage 1351
OleObjectInterface property
TOleContainer 1271
OnAccept event
TCustomSocket 623
OnActivate event
TApplication 29
TCustomForm 403
TOleContainer 1273
OnActiveControlChange event
TScreen 1428
OnActiveFormChange event
TScreen 1428
OnApply event
TFontDialog 1097
OnBeginPivot event
TDecisionSource 969
OnCalcFields event
TDataSet 718
OnCellClick event
TCustomDBGrid 325
OnChange event
TCoolBar 268
TCustomComboBox 287
TCustomEdit 382
TCustomImageList 466
TCustomListView 518
TCustomTabControl 641
TCustomTreeView 652
TDateTimePicker 773
TDBRadioGroup 889
TDdeClientItem 919
TDdeServerItem 928
TDirectoryListBox 991
TField 1056
TFileListBox 1070
TScrollBar 1434
TTrackBar 1596
OnChange method
TFilterComboBox 1077
OnChanging event
TCustomListView 518
TCustomTabControl 641
TCustomTreeView 652
TCustomUpDown 668
OnClick event
TControl 237
TCustomUpDown 668
TDBNavigator 881
TMediaPlayer 1208
OnClick method
TMenuItem 1243
OnclickCheck event
TCheckListBox 148
OnClientConnect event
TCustomServerSocket 608
OnClientDisconnect event
TCustomServerSocket 609
OnClientError event
TCustomServerSocket 610
OnClientRead event

TCustomServerSocket 611
OnClientWrite event
TCustomServerSocket 611
OnClose event
TAnimate 16
TCommonDialog 195
TCustomForm 403
TDdeClientConv 910
TDdeServerConv 922
OnCloseQuery event
TCustomForm 404
OnCloseUp event
TDateTimePicker 774
TDBLookupComboBox 848
OnColEnter event
TCustomDBGrid 326
OnColExit event
TCustomDBGrid 326
OnCollapsed event
TCustomTreeView 653
OnCollapsing event
TCustomTreeView 653
OnColumnClick event
TCustomListView 519
OnColumnMoved event
TCustomDBGrid 327
TDrawGrid 999
OnCompare event
TCustomListView 519
TCustomTreeView 653
OnConnect event
TCustomSocket 623
OnConnecting event
TCustomSocket 624
OnCreate event
TCustomForm 405
TDataModule 699
OnCreateContent event
TDSTableProducer 1018
OnDataChange event
TDataSource 752
OnDataRequest event
TCustomProvider 574
OnDblClick event
TControl 238
OnDeactivate event
TApplication 29
TCustomForm 405
TOleContainer 1274
OnDecisionDrawCell property
TCustomDecisionGrid 366
OnDecisionExamineCell property
TCustomDecisionGrid 367
OnDeleteError event
TDataSet 719
OnDeletion event
TCustomListView 519
TCustomTreeView 654
OnDestroy event
TCustomForm 406
TDataModule 699
OnDisconnect event
TCustomSocket 625
OnDragDrop event
TControl 238
OnDragOver event
TControl 239
OnDrawCell event
TDrawGrid 999
OnDrawColumnCell event
TCustomDBGrid 327
OnDrawDataCell event
TCustomDBGrid 328
OnDrawItem event
TCustomComboBox 287
TCustomListBox 496
OnDrawPanel event
TStatusBar 1505
OnDrawSection event
THeaderControl 1120
OnDropDown event
TCustomComboBox 288
TDateTimePicker 774
TDBLookupComboBox 849
OnEditButtonClick event
TCustomDBGrid 328
OnEdited event
TCustomListView 520
TCustomTreeView 654
OnEditError event
TDataSet 720
OnEditing event
TCustomListView 520
TCustomTreeView 654
OnEndDrag event
TControl 239
OnEndPivot event
TDecisionSource 970
OnEnter event
TWinControl 1641
OnError event
TCustomSocket 625
OnException event
TApplication 30
OnExecuteMacro event
TDdeServerConv 923
OnExit event
TWinControl 1641
OnExpanded event
TCustomTreeView 654
OnExpanding event
TCustomTreeView 655
OnFilterRecord event
TDataSet 720
OnFind event
TFindDialog 1083
OnFolderChange method
TOpenDialog 1296
OnFormatCell event
TDSTableProducer 1018
OnGetEditMask event
TDrawGrid 1000
OnGetEditText event
TDrawGrid 1000
OnGetImageIndex event
TCustomTreeView 655
OnGetSelectedIndex method
TCustomTreeView 655
OnGetSocket event
TCustomServerSocket 612
OnGetTableCaption event
TDSTableProducer 1019
OnGetText event
TField 1057
OnGetThread event
TCustomServerSocket 612
OnHelp event
TApplication 30
TCustomForm 406
OnHide event
TCustomForm 406
OnHint event
TApplication 31
OnHTMLTag event
TPageProducer 1310
OnIdle event
TApplication 32
OnInsert method
TCustomListView 520
OnKeyDown event
TWinControl 1642
OnKeyPress event
TWinControl 1643
OnKeyUp event
TWinControl 1643
OnLayoutChange event
TDecisionSource 970
online help *See* Help systems
OnListen event
TCustomSocket 625
OnLogin event
TDatabase 692
OnLookup event
TCustomSocket 626

OnLowCapacity event
TCustomDataStore 311
OnMeasureItem event
TCustomComboBox 288
TCustomListBox 498
OnMessage event
TApplication 32
OnMinimize event
TApplication 33
OnMouseDown event
TControl 240
OnMouseMove event
TControl 240
OnMouseUp event
TControl 241
OnMoved event
TSplitter class 1495
OnNewDimensions event
TDecisionSource 971
OnNewRecord event
TDataSet 721
OnNotify event
TMediaPlayer 1209
OnObjectMove event
TOleContainer 1274
OnOpen event
TAnimate 16
TDdeClientConv 910
TDdeServerConv 923
OnPaint event
TCustomForm 407
TPaintBox 1314
OnPaintPanel event
TDBCtrlGrid 799
OnPassword event
TSession 1462
OnPokeData event
TDdeServerItem 928
OnPopup event
TPopupMenu 1334
OnPostClick event
TMediaPlayer 1209
OnPostError event
TDataSet 721
OnProgress event
TImage 1138
OnProtectChange event
TCustomRichEdit 597
OnRead event
TCustomSocket 627
OnReconcileError event
TClientDataSet 163
OnRefresh event
TCustomDataStore 311
OnReplace event
TReplaceDialog 1394
OnResize Event
TToolBar 1577
OnResize event
TCoolBar 269
TCustomPanel 567
THeaderControl 1121
TOleContainer 1274
TScrollBox 1441
TStatusBar 1506
OnResize method
TCustomForm 407
OnResizeRequest event
TCustomRichEdit 597
OnRestore event
TApplication 33
OnRowMoved event
TDrawGrid 1000
OnSaveClipboard event
TCustomRichEdit 598
OnScroll event
TScrollBar 1435
OnSectionClick event
THeaderControl 1121
OnSectionResize event
THeaderControl 1121
OnSectionTrack event
THeaderControl 1122
OnSelectCell event
TDrawGrid 1001
OnSelectCell property
TCustomDecisionGrid 368
OnSelectionChange event
TCustomRichEdit 598
OnSelectionChange method
TOpenDialog 1297
OnServerYield event
TBDEDataSet 78
OnSetEditText event
TDrawGrid 1001
OnSetText event
TField 1057
OnShow event
TCommonDialog 196
OnShow method
TCustomForm 408
OnShowHint event
TApplication 34
OnStart event
TAnimate 16
OnStartDrag event
TControl 241
OnStartup event
TSession 1462
OnStateChange event
TDataSource 752
TDecisionSource 971
OnStop event
TAnimate 17
OnSummaryChange method
TDecisionSource 971
OnThreadEnd event
TCustomServerSocket 612
OnThreadStart event
TCustomServerSocket 613
OnTimer event
TTimer 1569
OnTitleClick method
TCustomDBGrid 328
OnTopLeftChanged event
TDrawGrid 1001
OnTopLeftChanged method
TCustomDecisionGrid 368
OnTypeChange method
TOpenDialog 1297
OnUpdateData event
TDataSource 753
OnUpdateError event
TBDEDataSet 79
TProvider 1357
OnUpdateRecord event
TBDEDataSet 80
OnUserInput event
TDateTimePicker 774
OnValidate event
TField 1057
OnWrite event
TCustomSocket 627
Open common dialogs 1291 to 1299
creating/destroying 1298
current directory 1294
default extensions 1292
file filters 1293, 1294
graphics files 1299 to 1302
reserved areas 1299
running 1299, 1302
selecting files 1292, 1293
setting defaults 1294
title bars 1296
Open method
TCustomSocket 630
TDatabase 696
TDataSet 741
TMediaPlayer 1217
TSession 1474
Open property
TAnimate 12
OpenCurrent method
TDirectoryListBox 995

OpenDatabase method
TSession 1474
OpenDimIndexLeft method
TDecisionSource 983
OpenDimIndexRight method
TDecisionSource 984
OpenIndexFile method
TTable 1555
opening
database sessions 1462, 1467, 1474
databases 697, 1474
datasets 741
OpenLink method
TDdeClientConv 914
option buttons *See* radio buttons
options 141, 275, 777
mutually exclusive 1378
Options property
TColorDialog 186
TCustomDBGrid 322
TCustomDecisionGrid 364
TCustomGrid 431
TFindDialog 1082
TFontDialog 1095
TOpenDialog 1294
TPrintDialog 1338
Oracle tables
overloaded procedures 1510
Orientation property
TCustomUpDown 666
TDBCtrlGrid 795
TTrackBar 1594
Origin property
TField 1052
origins 221, 251, 1633
list views 516
osEmpty constant 1272
osInPlaceActive constant 1273
osLoaded constant 1272
osOpen constant 1272
osRunning constant 1272
osUIActive constant 1273
output parameters *See* stored procedures
overlay masks 473, 478, 479
Overlay method
TCustomImageList 478
Overload property
TStoredProc 1510
owned components 202
counting 199
destroying 206, 207
getting 200, 207, 208, 1331
message processing 207, 212
renaming 216, 422
setting 211
Owner property
TComponent 202
owner-draw combo boxes 286, 787
event handling 287, 288
owner-draw list boxes 488
creating 495
displaying items 492, 496, 498, 504, 505
drawing surfaces 490

P

paCenter constant 1332
PacketRecords property
TClientDataSet 159
packets 153, 573
adding records 159, 576, 577
change logs 154, 158
getting 154, 174
loading resources 480
page producers 557, 1309
creating/destroying 561
defining templates 558, 562
event handling 1310
getting contents 560, 561
template files 558
page sets 1302 to 1309
See also tab sheets
accessing pages 1305, 1308, 1309
activating current page 1304
adding pages 1558
counting pages 1304
creating/destroying 1307, 1308
PageControl property
TTabSheet 1559
PageCount property
TPageControl 1304
PageIndex property
TTabSheet 1560
PageRect property
TCustomRichEdit 594
Pages property
TPageControl 1305
PageSize property
TTrackBar 1594
paint boxes 1312 to 1316
creating 1316
event handling 1314
Paint method
TCustomControl 302
TCustomDecisionGrid 374
TCustomForm 419
TCustomGrid 448
TCustomGroupBox 456
TCustomLabel 487
TCustomPanel 571
TGraphicControl 1109
TMediaPlayer 1217
TPaintBox 1316
TSplitter class 1498
PaintControls method
TWinControl 1665
PaintHandler method
TWinControl 1665
painting components
image lists 463, 464, 465, 469, 472, 473
painting controls 302, 1314
column headers 1120
combo boxes 293
data grids 320, 327, 328, 337, 339, 340, 796, 799
decision grids 373, 374
draw grids 1004
forms 407, 419
graphics 1109, 1316
grids 427, 443, 448
group boxes 456
images 833
labels 487
nonwindowed 252, 254, 257, 262
panels 567, 571
radio buttons 583
splitters 1498
windowed 1633, 1661, 1665, 1666, 1668, 1674
PaintTo method
TWinControl 1665
PaintWindow method
TCustomControl 302
TCustomForm 419
TWinControl 1666
paLeft constant 1332
PaletteChanged method
TControl 256
TCustomForm 419
TWinControl 1666
palettes 252, 833
realizing 256, 419, 1666
PanelBorder property
TDBCtrlGrid 796
PanelCount property
TDBCtrlGrid 796
PanelHeight property
TDBCtrlGrid 796

PanelIndex property
TDBCtrlGrid 796
panels 91, 562, 804, 1316, 1502
aligning controls 570
bevel styles 564, 565
borders 566, 796
captions 564
creating 95, 570, 571
data grids 796, 797
event handling 567
locking 567
painting 567, 571
resizing 567, 571
styles 93
Panels property
TStatusBar 1504
PanelWidth property
TDBCtrlGrid 797
Paradox tables 1546
accessing 1541
aliases 1466
locking 1555, 1557
network control files 1459
password-protecting 1462, 1466, 1472, 1475
querying 1364
working directories 684
Paragraph property
TCustomRichEdit 594
ParamBindMode property
TStoredProc 1511
ParamByName method
TQuery 1373
TStoredProc 1518
ParamCheck property
TQuery 1365
ParamCount property
TQuery 1365
TStoredProc 1512
parameterized queries 1365, 1366
parameters, input/output *See* stored procedures
Params property
TDatabase 687
TQuery 1366
TStoredProc 1512
parent components 215, 261
getting 209, 210
parent controls 232, 260, 421
aligning 219
colors 232
handles 1659, 1660
help hints 233
text attributes 233
parent objects 1257
Parent property
TControl 232
TMenuItem 1241
ParentColor property
TControl 232
ParentCtl3D property
TWinControl 1638
ParentFont property
TControl 233
ParentShowHint property
TControl 233
ParentWindow property
TWinControl 1638
paRight constant 1332
ParseInput property
TDateTimePicker 772
Password common dialogs 1321 to 1327
accepting passwords 1327
adding passwords 1326
changing passwords 1326
removing passwords 1327
PasswordChar property
TCustomEdit 380
passwords 380, 1321, 1462, 1466
getting 1472
removing 1475
Paste method
TOleContainer 1282
Paste Special OLE dialog box 1282
PasteFromClipboard method
TCustomEdit 389
TDBImage 838
PasteLink method
TDdeClientConv 914
PasteSpecialDialog method
TOleContainer 1282
pasting OLE objects 1268, 1282
pasting text 389
Pause method
TMediaPlayer 1217
PauseOnly method
TMediaPlayer 1218
pbByName constant 1511
pbByNumber constant 1511
Pen property
TShape 1477
pens 1477
Perform method
TControl 256
persistent objects 1327 to 1331
copying 1329
Picture property
TDBImage 833
TImage 1137
pictures 1137
See also images
getting images 833
loading 838
pixels 1427
PixelsPerInch property
TCustomForm 399
TScreen 1427
PlainText property
TCustomRichEdit 595
Play method
TAnimate 19
TMediaPlayer 1218
poDefault constant 400
poDefaultPosOnly constant 400
poDefaultSizeOnly constant 400
poDesigned constant 400
poDisablePrintToFile constant 1338
poHelp constant 1338
pointer (mouse) *See* cursors
pointers 202
data buffers 724
data providers 573, 589
run-time type information 1256, 1257
PokeData method
TDdeClientConv 914
TDdeServerItem 931
PokeDataLines method
TDdeClientConv 915
poking data *See* DDE applications
poNone constant 400
poPageNums constant 1338
poPrintToFile constant 1338
poPrintToFit constant 400
poProportional constant 400
PopulateOle2Menu method
TMainMenu 1186
pop-up menus 234, 1331
See also menu objects; menus
alignment 1332
creating/destroying 255, 1335
displaying 1333, 1336
event handling 1334
OLE objects 1268
online help for 1333
returning active object 1333
testing for 252
toolbars 1582

Popup method
TPopupMenu 1336
PopupComponent property
TPopupMenu 1333
PopupMenu property
TControl 234
port IDs (sockets) 622, 631
Port property
TCustomSocket 622
poScreenCenter constant 400
poSelection constant 1338
PosEqual method
TInplaceEdit 1152
Position property
TCustomForm 399
TCustomUpDown 667
TFindDialog 1083
TMediaPlayer 1203
TProgressBar 1345
TScrollBar 1433
TTrackBar 1594
Post method
TDataSet 741
PostClick method
TMediaPlayer 1217
posting records 726, 741
poWarning constant 1338
prAllPages constant 1339
Precision property
TFloatField 1091
Prepare method
TQuery 1373
TStoredProc 1519
Prepared property
TQuery 1366
TStoredProc 1512
PreserveCase property
TDirectoryListBox 990
Preview property
TReport 1400
Previous method
TForm 1104, 1105
TMediaPlayer 1219
primary indexes 1548
multi-tiered applications 158
primary keys 46
primary windows *See* main windows
PrimaryVerb property
TOleContainer 1271
Print common dialogs 1336 to 1341
collating 1337
number of copies 1337
page ranges 1337, 1338, 1339
printing to files 1339
running 1341
setting defaults 1338
Print method
TCustomForm 420
TCustomRichEdit 604
TReport 1405
PrintCopies property
TReport 1400
Printer Setup common dialogs 1341 to 1343
running 1343
printing
forms 400, 420
reports 1398, 1401, 1405
multiple copies 1400
text 594, 604
printing conventions 2
printing options
collating 1337
number of copies 1337
page ranges 1337, 1338, 1339
sending to files 1339
PrintRange property
TPrintDialog 1339
PrintScale property
TCustomForm 400
PrintToFile property
TPrintDialog 1339
Prior method
TDataSet 741
PrivateDir property
TSession 1460
ProblemCount property
TBatchMove 57
Problems tables 52, 57
ProblemTableName property
TBatchMove 57
ProcessMessages method
TApplication 44
producers
content 295
page 557, 1309
property editors 1016
table 744, 1013, 1374
programming 1
programs *See* applications
progress bars 1343 to 1349
creating 1348
current position 1345
data stores 309
increments 1345
moving indicators 1345, 1348, 1349
progress indicators *See* progress bars
properties 1, 205
assigning values 257, 1667
data grids 322
forms 415, 420
grids 431, 442
image lists 471, 477
list views 524
OCX controls 1288, 1290
OLE objects 1282
publishing 248, 420, 1330
saving 230
property editors 1016
property pages 1349 to 1355
changing objects 1355
control references 1351
counting 1423
undoing changes 1355
updating 1355
protecting data *See* passwords
provider interfaces 160, 573, 1389
creating 577
Provider property
TClientDataSet 160
TCustomProvider 573
ProviderName property
TClientDataSet 160
providers 572 to 577, 1356 to 1361
applying updates 575, 1360
creating 1360
creating packets 153, 573
destroying 576
error handling 1357
event handling 574, 576, 1357
getting 589
getting data 1357, 1360, 1361
handling client requests 1357, 1360, 1361
naming 160
receiving client changes 154, 158
retrieving records 159, 576, 577
returning 589, 1392
sending packets 154, 174
prPageNums constant 1339
prSelection constant 1339
published objects 1260
publishing 248, 420
push buttons 100, 125
See also buttons

creating/destroying 130
default 127

Q

queries 958, 1361
See also SQL statements
accessing tables 1364
cached updates 1607, 1608, 1609, 1610
counting parameters 1365
creating 1372
creating statements 1367
data sources 1364
generating parameter lists 1365
getting parameter values 1366, 1373
getting statements 1368
HTTP documents 1374 to 1378
read-only datasets 1604
result sets 1363, 1366, 1369
running 1366, 1373, 1374, 1610
subsets of values 935
tracking changes 1367
updating records 747
Query property
TQueryTableProducer 1376
TUpdateSQL 1607
QueryInterface method
TComponent 212
QuickDraw property
TDBImage 833

R

raAbort constant 164
raCancel constant 164
raCorrect constant 164
radio buttons 1378 to 1383
adding 1383
captions 887, 1380
changing values 889
columns 579
getting values 583, 886, 887, 888
grouping 578, 884, 1383
iterating through 580
read-only 887
selecting 579, 582, 891, 1380
RadioItem property
TMenuItem 1242
raMerge constant 164
raRefresh constant 164
raSkip constant 164
RawToDataColumn method
TCustomDBGrid 346
read-only components 1604
datasets 161
field objects 1053
tables 1545
read-only controls 597
check boxes 779
combo boxes 787
data grids 323
edit boxes 381, 818, 870, 895
images 833
list boxes 841, 862
list views 513
radio buttons 887
tree views 649
read-only files 1068
ReadOnly property
TClientDataSet 161
TCustomDBGrid 323
TCustomEdit 381
TCustomListView 513
TCustomTreeView 649
TDBCheckBox 779
TDBComboBox 787
TDBEdit 818
TDBImage 833
TDBListBox 841
TDBLookupControl 856
TDBLookupListBox 862
TDBMemo 870
TDBRadioGroup 887
TDBRichEdit 895
TField 1053
TTable 1545
ReadState method
TComponent 212
TControl 257
TCustomForm 420
TCustomRadioGroup 583
TWinControl 1667
Ready property
TDecisionSource 968
Realign method
TWinControl 1667
rebars 263 to 271
adding text 267
alignment 265, 268
band collections 266
borders 266
creating/destroying 271
event handling 268, 269
images 266, 267
reordering bands 267
size 265, 267
RecalcReport method
TReport 1405
RecNo property
TBDEDataSet 75
TClientDataSet 161
TDataSet 710
Reconcile method
TClientDataSet 178
RecordCount property
TBatchMove 58
TBDEDataSet 75
TClientDataSet 161
TDataSet 711
records 804
See also data; databases; tables
activating 743, 797
adding 737, 794, 1606
appending 167, 725
batch processing 50, 1549
canceling changes 85, 726
changing 710
counting 46, 75, 161, 711, 796, 797
current 75, 88, 710, 735, 796
deleting 729, 793, 1606
displaying 77
fetching 88
finding 710
free-form layouts 791
locating specific 89, 739
moving through 734, 738, 739, 740, 741
moving to specific 1554
packaging 159, 576, 577
position 704, 706, 733, 734
posting 726, 741
reconciling updates 178
restoring previous values 178, 181
size 76, 161, 711, 796, 797
synchronizing 742, 1554
testing existence 738
undoing changes 90, 178, 181
updating 91, 744, 747, 1607
RecordSize property
TBDEDataSet 76
TClientDataSet 161
TDataSet 711
RecreateWnd method
TWinControl 1668
rectangles 1478
redrawing *See* painting
references 201, 217

ActiveX controls 1351
COM interfaces 207
controls 232
ReformatText method
TCustomMaskEdit 546
Refresh method
TControl 257
TCustomDataStore 316
TDataSet 742
RefreshLookupList
TField 1063
RegisterChanges method
TCustomImageList 479
RegisterConversionFormat method
TCustomRichEdit 604
registering
components 215
_Release method
TComponent 217
Release method
TCustomForm 420
ReleaseHandle method
THintWindow 1130
remote applications
getting data 584, 587
removing data 590
remote connections 583, 687, 1388
application servers 588, 1389, 1391, 1392
data links 584, 587, 590
dropping 590, 1391
testing 584, 588
remote servers 162, 583, 1388
creating/destroying 587, 588
data providers 160, 589, 1392
event handling 585, 586
loading 589
names 1389, 1390
provider interfaces 1389
remote SQL servers 689
RemoteServer property
TClientDataSet 162
Remove method
TMenuItem 1247
RemoveAllButtonClick method
TPasswordDialog 1327
RemoveAllPasswords method
TSession 1475
RemoveButtonClick method
TPasswordDialog 1327
RemoveComponent method
TComponent 213
RemoveControl method
TWinControl 1668
RemoveDataSet method
TCustomRemoteServer 590
RemovePassword method
TSession 1475
removing *See* deleting
RenameTable method
TTable 1555
renaming components 204, 216
renaming tables 1555
Repaint method
TControl 257
TWinControl 1668
repainting *See* painting
Repetitions property
TAnimate 13
Replace common dialogs 1393 to 1396
creating 1396
replacement strings 1394
Replace method
TCustomImageList 479
ReplaceIcon method
TCustomImageList 479
ReplaceMasked method
TCustomImageList 479
ReplaceText property
TReplaceDialog 1394
replacing text 1393
ReportBasic macros 1406
ReportDir property
TReport 1400
ReportHandle property
TReport 1400
ReportName property
TReport 1401
reports 1396
See also ReportSmith applications
ReportSmith applications 1396
closing 1403
connecting to servers 1404
creating/destroying 1404, 1405
executing macros 1406
getting records 1399
getting versions 1401, 1402
initializing report variables 1398
launching 1399
previewing reports 1400
printing reports 1398, 1401, 1405
multiple copies 1400
quitting 1397
recalculating reports 1405
running reports 1401, 1406
saving reports 1400
setting variables 1407
status 1398
request messages *See* HTTP
Request property
TCustomWebDispatcher 674
RequestData method
TDdeClientConv 915
RequestLive property
TQuery 1366
RequestSize method
TCustomRichEdit 605
Required property
TField 1053
Reset method
TAnimate 20
TCustomMaskEdit 546
TCustomProvider 577
TDBEdit 824
TProvider 1361
ResetContent method
TCustomListBox 506
ResetIme method
TWinControl 1669
ResetImeComposition method
TWinControl 1669
ResHandle property
TAnimate 13
ResID property
TAnimate 13
resizable column headers 1118
Resize method
TCustomForm 420
TCustomPanel 571
resizing controls 245
resizing forms 407, 420, 1493
ResName property
TAnimate 14
resource IDs 13, 195
ResourceLoad method
TCustomImageList 480
resources
image lists 476
loading
images 473, 474, 480, 1136
pictures 838
response messages *See* HTTP
Response property
TCustomWebDispatcher 675
Restore method
TApplication 44
RestoreTopMosts method
TApplication 44

result sets 1363, 1366
bidirectional cursors 1369
Resume method
TMediaPlayer 1219
Resync method
TDataSet 742
return characters 551, 556
RevertRecord method
TBDEDataSet 90
TClientDataSet 178
Rewind method
TMediaPlayer 1220
rich text boxes *See* rich text edit controls
rich text edit controls 590 to 606, 891, 1408
attributes 592, 595
binding to data fields 894, 895
changing contents 898
clearing text 601
conversions 593, 595, 604
creating/destroying 601, 602, 899
displaying 593
event handling 597 to 598
formatting 594
getting selections 603, 604
getting text 594, 603, 894, 895, 900
keyboard processing 899, 900
message processing 900
printing 594, 604
read-only 597
saving 595, 598
scrolling 593
selecting text 596, 598, 605, 606
size 597, 603, 605
rich text files 604
Right-click menus *See* context menus
Rollback method
TDatabase 697
rollbacks *See* database transactions
rounded rectangles 1478
rounded squares 1478
Row property
TCustomGrid 432
RowAttributes property
TDSTableProducer 1017
RowCount property
TCustomDecisionGrid 365
TCustomGrid 432
TDBCtrlGrid 797
TDBLookupListBox 862
TToolBar 1576
RowHeight property
TCustomDecisionGrid 365
RowHeights method
TCustomGrid 432
RowHeightsChanged method
TCustomDecisionGrid 375
TCustomGrid 449
RowMoved method
TCustomGrid 449
TDrawGrid 1005
rows
See also grids; records
decision grids 362, 365, 375
list boxes 862
list views 517
table producers 1017
Rows property
TStringGrid class 1530
RowsAffected property
TQuery 1367
RowSelect property
TCustomListView 514
rrAbort constant 1358
rrApply constant 1358
rrCorrect constant 1358
rrIgnore constant 1358
rrMerge constant 1358
rrSkip constant 1358
rtDeleted constant 77
rtInserted constant 77
rtModified constant 77
RTTI *See* run-time type information
rtUnmodified constant 77
Run method
TApplication 45
TOleContainer 1282
TReport 1406
TWebApplication 1626
Run property
TCGIApplication 140
RunMacro method
TReport 1406
running batch moves 60, 61
run-time type information 1256, 1257

S

SafeCallException method
TComponent 213
TObject 1264
Save As common dialogs 1415 to 1417
Save common dialogs 1413 to 1415
Save method
TMediaPlayer 1220
SaveConfigFile method
TSession 1475
SaveSeries property
TCustomDecisionGraph 351
SaveToFile method
TBlobField 118
TClientDataSet 179
TCustomTreeView 662
TOleContainer 1283
SaveToStream method
TBlobField 118
TClientDataSet 179
TCustomTreeView 662
TOleContainer 1283
saving client datasets 179
saving files 1413
saving property settings 230
saving reports 1400
sbHorizontal constant 1432
sbsNone constant 634
sbsSingle constant 634
sbsSunken constant 634
sbVertical constant 1432
ScaleBy method
TWinControl 1670
ScaleControls method
TWinControl 1670
Scaled property
TCustomForm 401
scaling constants 234
ScalingFlags property
TControl 234
scBottom constant 1435
scEndScroll constant 1435
scLineDown constant 1435
scLineUp constant 1435
scPageDown constant 1435
scPageUp constant 1435
scPosition constant 1435
screen coordinates
client areas 246, 251, 258
conversions 246, 258
grids 438, 439
list views 516
origins 221, 251, 1633
screen objects 1418 to 1430
creating/destroying 1429, 1430
cursor constants 1420

event handling 1428
fonts 1425
getting active control 1419
getting opened forms 1419, 1420, 1423, 1425
size 1426, 1427
vertical measurements 1427
Screen variable 1418
ScreenToClient method
TControl 258
scroll bars 1430
See also scrolling
adding 550, 557, 1432, 1445
creating/destroying 1437
event handling 1434, 1435
hiding 593
moving 1432, 1434
position 1433, 1438, 1445
scroll boxes 1438 to 1443
creating 1443
event handling 1441
resizing 1441
Scroll method
TCustomDBGrid 346
TCustomListView 532
scrollable lists 145, 189, 281, 488, 1167
ScrollBars method
TCustomGrid 433
ScrollBars property
TCustomMemo 550
ScrollBy method
TWinControl 1671
ScrollData method
TCustomGrid 449
scrolling
See also scroll bars; scroll boxes
data grids 346
decision grids 375
edit controls 550, 557, 593
grids 428, 433, 449, 451, 452
list views 532
windowed controls 1443, 1671
ScrollInView method
TScrollingWinControl 1451
ScrollOpposite property
TCustomTabControl 639
scTop constant 1435
scTrack constant 1435
sdAbove constant 531
sdAll constant 531
sdBelow constant 531
sdLeft constant 531
sdRight constant 531
searching 157, 1080
common dialogs 1393
datasets 90, 710, 732, 733, 739
filtered records 733, 734
multi-tiered
applications 171, 173, 176
specific records 89, 739
list views 532
rich text edit controls 603
tables 1551
specified values 1552, 1553
secondary indexes 1543
deleting 1550
Sections property
THeaderControl 1120
security 1177, 1321
Seek method
TAnimate 20
SelAttributes property
TCustomRichEdit 595
SelCount property
TCustomListBox 494
TCustomListView 514
SelectAll method
TCustomComboBox 294
TCustomEdit 389
SelectCell method
TCustomDecisionGrid 375
TCustomGrid 450
TDrawGrid 1006
Selected property
TCustomListBox 494
TCustomListView 514, 650
SelectedColor property
TDBCtrlGrid 797
SelectedField property
TCustomDBGrid 323
SelectedIndex property
TCustomDBGrid 323
SelectedItem property
TDBLookupListBox 862
SelectedRows property
TCustomDBGrid 324
SelectFirst method
TWinControl 1671
selecting text
edit controls 378, 379, 381, 385, 389, 390
masked edit controls 545, 547
rich text edit boxes 596, 598, 605, 606
Selection method
TCustomGrid 433
SelectionChange method
TCustomRichEdit 605
SelectNext method
TWinControl 1671
SelectNextPage method
TPageControl 1309
SelEnd property
TTrackBar 1594
SelLength property
TCustomComboBox 285
TCustomEdit 381
TCustomRichEdit 596
SelStart property
TCustomComboBox 285
TCustomEdit 381
TCustomRichEdit 596
TTrackBar 1595
SelText property
TCustomComboBox 286
TCustomEdit 381
TCustomRichEdit 596
SendCancelMode method
TControl 258
TCustomForm 421
SendToBack method
TControl 259
SerVariableLines method
TReport 1407
server applications 925
connections 906, 921, 927
enabling/disabling constraints 87, 171, 172
Web sites 138, 1160, 1619
server connections 162, 1389, 1391, 1392
server sockets 606, 1451
connecting to 620, 629, 630
creating 1454
destroying 615
disconnecting from 629
IP addresses 621, 631
listening endpoints 1452
port IDs 622, 631
standard services 622, 631
threads 607, 615, 616, 617
setting 617, 618, 619
types 607, 616, 619
ServerConv property
TDdeServerItem 927
ServerName property
TRemoteServer 1390
ServerType property
TCustomServerSocket 607

Service property
TCustomSocket 622
ServiceApplication property
TDdeClientConv 909
Session property
TDatabase 688
SessionAlias property
TDatabase 688
SessionName property
TDatabase 688
TSession 1460
sessions 688, 1454
aliases 688
BDE aliases 1456, 1470, 1471
adding 1464, 1466
changing 1474
removing 1468
testing for 1473
closing 1466, 1467, 1468
configuration
parameters 1456, 1471
saving 1475
counting active
databases 1457
event handling 1462
getting active
databases 1458, 1469, 1471
getting stored
procedures 1472
getting table names 1473
language drivers 1459, 1465, 1470, 1471, 1472
removing 1468
maintaining
connections 1459
monitoring SQL
operations 1461
naming 1456, 1460
network control files 1459
opening databases 1474
removing passwords 1475
starting 1462, 1467, 1474
storing temporary
tables 1460
terminating
connections 1467, 1469
testing for active 1455
SetActive method
TCustomSocket 630
SetAddress method
TCustomSocket 631
SetAlignment method
TCustomMemo 556
SetAncestor method
TComponent 213
SetAutoSize method
TCustomLabel 487
SetBounds method
TControl 259
TDBNavigator 884
TDecisionPivot 958
TOleControl 1290
TWinControl 1672
SetChildOrder method
TComponent 214
TCustomForm 421
TWinControl 1672
SetColumnAttributes method
TCustomDBGrid 346
SetColumns method
TDSTableProducer 1026
SetConnected method
TCustomRemoteServer 590
SetCurrentSummary method
TDecisionSource 984
SetCursor method
TCustomMaskEdit 547
SetData method
TField 1064
SetDataSet method
TDSTableProducer 1026
SetDataSource method
TDSTableProducer 1026
SetDesigning method
TComponent 214
SetDragImage method
TCustomImageList 480
SetDragMode method
TControl 260
SetEditText method
TCustomDBGrid 347
TCustomGrid 450
TDrawGrid 1006
SetFields method
TDataSet 743
SetFieldType method
TBlobField 118
TField 1064
SetFocus method
TCustomForm 421
TInplaceEdit 1152
TWinControl 1672
SetFocusedControl method
TCustomForm 421
SetGetSocketEvent method
TCustomServerSocket 617
SetGetThreadEvent method
TCustomServerSocket 617
SetHost method
TCustomSocket 631
SetIme method
TWinControl 1673
SetImeCompositionWindow method
TWinControl 1673
SetItemData method
TCustomListBox 506
SetKey method
TClientDataSet 179
TTable 1556
SetLines method
TCustomMemo 556
SetLink method
TDdeClientConv 916
SetName method
TComponent 214
TControl 260
SetOle2MenuHandle method
TMainMenu 1186
SetOnClientError method
TCustomServerSocket 618
SetOnClientEvent method
TCustomServerSocket 618
SetOnThreadEnd method
TCustomServerSocket 618
SetOnThreadStart method
TCustomServerSocket 618
SetParams method
TScrollBar 1438
TUpdateSQL 1610
SetParent method
TControl 260
TCustomForm 421
SetParentComponent method
TComponent 215
TControl 261
SetPort method
TCustomSocket 631
SetPropDisplayString method
TOleControl 1291
SetRange method
TClientDataSet 179
TTable 1556
SetRangeEnd method
TClientDataSet 180
TTable 1557
SetRangeStart method
TClientDataSet 180
TTable 1557
SetScrollBars method
TCustomMemo 557
SetSel method
TCustomMaskEdit 547
SetSelLength method
TCustomEdit 389

TCustomRichEdit 605
SetSelStart method
TCustomEdit 390
TCustomRichEdit 606
SetSelTextBuf method
TCustomEdit 390
SetServerType method
TCustomServerSocket 619
SetService method
TCustomSocket 631
SetStyle method
TCustomComboBox 294
SetText method
TBlobField 119
SetTextBuf method
TControl 261
SetThreadCacheSize method
TCustomServerSocket 619
SetTick method
TTrackBar 1599
settings *See* properties
SetVariable method
TReport 1407
SetWordWrap method
TCustomMemo 557
SetZOrder method
TControl 261
TWinControl 1673
sfFont constant 234
sfHeight constant 234
sfLeft constant 234
sfTop constant 234
sfWidth constant 234
Shape property
TBevel 92
TShape 1478
shapes 1476 to 1480
brushes 1477
creating/destroying 1480
pens 1477
styles 1478
Shareable property
TMediaPlayer 1203
ShareImages property
TCustomImageList 466
Shift-state constants 1642, 1643, 1662, 1663
shortcut keys *See* accelerators
ShortCut property
TMenuItem 1242
Show method
TControl 262
TCustomForm 422
ShowAboutBox method
TOleControl 1291
ShowAccelChar property
TCustomLabel 484
TCustomStaticText 634
ShowButtons property
TCustomTreeView 650
ShowCaptions property
TToolBar 1576
ShowCheckbox property
TDateTimePicker 772
ShowColumnHeaders property
TCustomListView 515
ShowControl method
TWinControl 1674
ShowCubeDialog method
TDecisionCube 936
ShowDragImage method
TCustomImageList 480
ShowEditor method
TCustomGrid 450
ShowEditorChar method
TCustomGrid 450
ShowException method
TApplication 45
ShowFocus property
TDBCtrlGrid 797
ShowGlyph property
TFileListBox 1069
ShowHint property
TApplication 27
TControl 235
Showing property
TWinControl 1639
ShowLines property
TCustomTreeView 650
ShowMainForm property
TApplication 27
ShowModal method
TCustomForm 422
ShowProgressDialog property
TCustomDataStore 309
ShowRoot property
TCustomTreeView 651
ShowText property
TCoolBar 267
SimplePanel property
TStatusBar 1504
SimpleText property
TStatusBar 1505
single-line edit controls 815, 1027
single-tierred applications 150
Size property
TField 1054
SizeChanged method
TCustomGrid 451
SizeGrip property
TStatusBar 1505
SizeMode property
TOleContainer 1271
Sizing method
TCustomGrid 451
sliders 1591
SmallChange property
TScrollBar 1434
SmallImages property
TCustomListView 515
SmallInt field types 1481 to 1485
creating 1484
smAutoSize constant 1272
smCenter constant 1272
smClip constant 1272
smScale constant 1272
smStretch constant 1272
Socket property
TClientSocket 183
TServerSocket 1452
sockets 606, 619, 1451
clients 181
connections 183, 620, 629, 630
closing 629
opening 630
server types 607, 616, 619
creating 1454
destroying 615
error handling 615, 625, 629
event handling 623 to 627, 629
servers 608 to 613, 615, 616, 617, 618
host names 621, 631
IP addresses 621, 631
aliases 621
listening endpoints 1452
server port IDs 622, 631
standard services 622, 631
threads 607, 615, 616, 617
setting 617, 618, 619
Sorted property
TCustomComboBox 286
TCustomListBox 494
sorting
list views 515, 519, 523, 527
tree views 651, 653, 658
SortType property
TCustomListView 515
TCustomTreeView 651
sound effects *See* media players
source code 1

Source property
TBatchMove 58
source tables 58
SourceDoc property
TOleContainer 1272
Spacing property
TBitBtn 105
TSpeedButton 1490
SparseCols property
TDecisionSource 969
SparseRows property
TDecisionSource 969
speed buttons 1485 to 1492
adding images 1487, 1489
borders 1487
captions 1489, 1490
creating/destroying 1492
enabling/disabling 1487
grouping 1488
placing images 1489
selecting 1486, 1492
SpeedMenus *See* context menus
spin boxes *See* up-down controls
splitters 1493 to 1498
creating 1497
event handling 1495, 1497, 1498
painting 1498
size 1494
SQL Links driver 686
SQL Monitor 689, 1461
SQL property
TQuery 1367
TSession 1461
TUpdateSQL 1608
SQL servers 689
SQL statements 958, 1361
See also queries
accessing tables 1364
cached deletions/insertions 1606, 1607, 1609
counting parameters 1365
creating 1367, 1372
data sources 1364
generating parameter lists 1365
getting parameter values 1366, 1373
monitoring 1461
read-only datasets 1604
result sets 1363, 1366, 1369
returning 1368
running 1366, 1373, 1374, 1610
subsets of values 935
tracking changes 1367
updating records 747
SQL tables 1472
aliases 1464
language drivers 1465
schema information 696
updating 1607, 1608, 1609, 1610
SQLBinary property
TQuery 1368
squares 1478
ssAlt constant 1643, 1644, 1663, 1664
ssBoth constant 433, 550
ssCtrl constant 1643, 1644, 1663, 1664
ssDouble constant 1643, 1644, 1663, 1664
ssHorizontal constant 433, 550
ssLeft constant 1643, 1644, 1663, 1664
ssMiddle constant 1643, 1644, 1663, 1664
ssNone constant 433, 550
ssShift constant 1643, 1644, 1663, 1664
ssVertical constant 433, 550
Start property
TMediaPlayer 1204
StartAdvise method
TDdeClientConv 916
StartFrame property
TAnimate 14
starting database sessions 1462, 1467, 1474
StartPage property
TReport 1401
StartPos property
TMediaPlayer 1204
StartRecording method
TMediaPlayer 1220
StartTransaction method
TDatabase 697
state constants 200
check boxes 147, 277
combo boxes 287
controls 224
datasets 711, 751
decision sources 981
list views 531
masked edit controls 538
OLE objects 1272
State event
TOleContainer 1272
State property
TCheckListBox 147
TCustomCheckBox 277
TDataSet 711
TDataSource 751
StateImages property
TCustomListView 516
TCustomTreeView 651
static text controls 481, 632, 1498
accelerators 634
alignment 633
borders 634
creating 636
receiving focus 634
resizing 633
static text fields 597
multiline 870
status bars 1502 to 1508
creating/destroying 1508
displaying information 1504, 1505
drawing surfaces 1504
event handling 1505, 1506
size 1505
stBoth constant 516, 651
stCircle constant 1478
stData constant 515, 651
stEllipse constant 1478
Step method
TMediaPlayer 1221
Step property
TProgressBar 1345
StepBy method
TProgressBar 1348
StepIt method
TProgressBar 1349
StmtHandle property
TQuery 1368
TStoredProc 1513
stNone constant 515, 651
Stop method
TAnimate 20
TMediaPlayer 1221
StopFrame property
TAnimate 15
StopSizing method
TSplitter class 1498
StorageInterface event
TOleContainer 1273
StoreColumns method
TCustomDBGrid 347
TDSTableProducer 1027
stored procedures 1508 to 1519
See also queries
copying parameters 1516

counting parameters 1512
creating/destroying 1517
executing 1512, 1518, 1519
getting names 1472, 1513
overloaded 1510
parameter information 1517, 1518
parameter list assignment 1511
returning parameter values 1518
storing parameters 1512
StoredProcName property
TStoredProc 1513
storing data
binary decimals 67
streams 1327
stRectangle constant 1478
Stretch property
TDBImage 834
TImage 1137
string fields 1519 to 1526
creating 1526
storing values 1524
string grids 423, 1526 to 1533
See also grids
accessing data 1529, 1530
associating objects with 1530
creating/destroying 1533
string types 1257
strings 1037
assigning values 1521, 1522, 1523, 1524
comparing 74
conversions 65, 112, 121, 763, 1089, 1156, 1524
StringWidth method
TCustomListView 532
stRoundRect constant 1478
stRoundSquare constant 1478
stSquare constant 1478
stText constant 516, 651
style constants
beveled panels 93
borders 4, 394, 634, 648, 951, 1440
buttons 106
combo boxes 286, 787
controls 225
forms 396
geometric shapes 1478
images 464, 831
list boxes 495, 861
list views 517
OLE objects 1268
panels 564, 565, 566
toolbars 1583
track bars 1596
Style property
TBevel 93
TBitBtn 106
TCustomComboBox 286
TCustomListBox 495
TDBComboBox 787
TToolButton 1583
subclassing 1651
subdirectories *See* directory list boxes
submenus *See* menu items; menus
substrings *See* strings
subtotals 361, 364, 365
summary values 303, 305, 348, 937, 965
See also decision grids
as variant 314, 316, 975, 976, 980
choosing 949
color options 360
displaying 361
getting active number 968
setting 984
SummaryCount property
TCustomDataStore 309
SwapDimIndexes method
TDecisionSource 985
system colors 223
system files 1068
system messages 263, 1675

T

tab controls 637, 1533, 1558
See also tab sets
activating tabs 638
adding tabs 640
aligning 644
client areas 638
creating/destroying 645
event handling 641
getting specific tab 640
multiline 639
selecting tabs 638, 641, 644
size 639, 641
tab order 1639, 1658, 1660
data grids 803
moving through 1671
tab sets 1533
See also tab controls
tab sheets 1558 to 1563
See also page sets
accessing 1560
creating/destroying 1563
displaying 1560
tab stops
grids 433
memos 551
TabHeight property
TCustomTabControl 639
TabIndex property
TCustomTabControl 640
TTabSheet 1560
table producers 744, 1013, 1374
adding rows 1017
captions 1014, 1023
columns 1014, 1021, 1024, 1026
changing 1025
storing 1027
creating/destroying 1022
data links 1015, 1016, 1025
display attributes 1017
event handling 1018, 1019
field mappings 1022
formatting data 1023
getting data 745, 747, 1015, 1024, 1025, 1375, 1376
headers and footers 1016
layout locks 1021
message processing 1026
producing HTML commands 1023
property editors 1016
running queries 1377
setting contents 1026
updating contents 1021, 1024
TableAttributes property
TDSTableProducer 1017
TableLevel property
TTable 1545
TableName property
TTable 1546
tables 1538, 1544
accessing 1541
adding records 1549
canceling data ranges 1549
constraints 1540
creating/destroying 1550, 1551
data sources 1540, 1544
defining indexes 1542, 1543, 1548
deleting 1551
emptying 1552
getting data ranges 1548, 1551, 1552

getting index
information 1541, 1553
getting names 1473
locking 1555, 1557
master-detail
relationships 1544
modifying 1540
moving to specific
records 1554
naming/renaming 1546,
1555
read-only 1545
removing secondary
indexes 1550
restructuring 1550
returning data ranges 1543,
1544
searching 1551, 1552, 1553
setting data ranges 1556,
1557
synchronizing current
records 1554
types 1546
TableType property
TTable 1546
TabOrder property
TWinControl 1639
Tabs property
TCustomTabControl 640
tabs *See* tab controls; tab sets
TabStop property
TWinControl 1640
TabStops method
TCustomGrid 433
tabular grids *See* data grids
TabVisible property
TTabSheet 1560
TabWidth property
TCustomListBox 495
TCustomTabControl 641
taCenter constant 550, 633
field objects 1035
labels 483
panels 564
TActiveForm 3 to 8
instantiating 7
Tag property
TComponent 202
taLeftJustify constant 550, 633
field objects 1035
labels 482
panels 564
radio buttons 1380
TAnimate 8 to 20
instantiating 19
TApplication 21 to 46
instantiating 38
taRightJustify constant 550, 633
field objects 1035
labels 483
panels 564
radio buttons 1380
TAutoIncField 46 to 50
instantiating 49
TBatchMove 50 to 61
instantiating 60
TBCDField 61 to 70
instantiating 69
TBDEDataSet 70 to 91
instantiating 86
TBevel 91 to 95
instantiating 95
tbHorizontal constant 401, 1594
TBinaryField 95 to 100
instantiating 99
TBitBtn 100 to 109
instantiating 109
TBlobField 109 to 119
instantiating 117
TBooleanField 119 to 125
instantiating 125
tbsButton constant 1583
tbsCheck constant 1583
tbsDivider constant 1583
tbsDropDown constant 1583
tbsSeparator constant 1583
TButton 125 to 131
instantiating 130
TButtonControl 131 to 134
tbVertical constant 401, 1594
TBytesField 134 to 138
instantiating 138
TCGIApplication 138 to 140
TCheckBox 141 to 143
TCheckListBox 145 to 150
TClientDataSet 150 to 181
instantiating 170
TClientSocket 181 to 185
instantiating 184
tcLowerCase constant 1009
TColorDialog 185 to 189
instantiating 188
TComboBox 189 to 193
TCommonDialog 193 to 197
instantiating 197
TComponent 198 to 217
instantiating 204
TControl 217 to 263
instantiating 247
TCoolBar 263 to 271
instantiating 271
TCP/IP applications 619
clients 181 to 185
servers 606, 1451
tcUpperCase constant 1009
TCurrencyField 271 to 275
instantiating 275
TCustomCheckBox 275 to 280
instantiating 280
TCustomComboBox 281 to 294
instantiating 292
TCustomContentProducer 295
to 298
TCustomControl 298 to 303
instantiating 302
TCustomDataStore 303 to 317
instantiating 313
TCustomDBGrid 317 to 348
instantiating 335
TCustomDecisionGraph 348 to
357
instantiating 356
TCustomDecisionGrid 357 to
375
instantiating 372
TCustomEdit 376 to 390
instantiating 386
TCustomForm 390 to 423
instantiating 412
TCustomGrid 423 to 452
instantiating 440
TCustomGroupBox 452 to 456
instantiating 456
TCustomHotKey 456 to 462
instantiating 461
TCustomImageList 462 to 481
instantiating 470
TCustomLabel 481 to 488
instantiating 486
TCustomListBox 488 to 507
instantiating 501
TCustomListView 507 to 534
instantiating 526
TCustomMaskEdit 534 to 548
instantiating 543
TCustomMemo 548 to 557
instantiating 554
TCustomPageProducer 557 to
562
instantiating 561
TCustomPanel 562 to 571
instantiating 570
TCustomProvider 572 to 577
TCustomRadioGroup 578 to
583

instantiating 582
TCustomRemoteServer 583 to 590
instantiating 587
TCustomRichEdit 590 to 606
instantiating 601
TCustomServerSocket 606 to 619
TCustomSocket 619 to 631
TCustomStaticText 632 to 636
instantiating 636
TCustomTabControl 637 to 646
instantiating 645
TCustomTreeView 646 to 662
instantiating 658
TCustomUpDown 663 to 672
instantiating 671
TCustomWebDispatcher 673 to 680
instantiating 677
TDatabase 680 to 697
instantiating 695
TDataModule 698 to 701
instantiating 699, 701
TDataSet 702 to 744
instantiating 728
TDataSetTableProducer 744 to 747
TDataSetUpdateObject 747 to 749
TDataSource 749 to 755
instantiating 754
TDateField 755 to 759
instantiating 759
TDateTimeField 760 to 768
instantiating 768
TDateTimePicker 768 to 777
instantiating 776
TDBCheckBox 777 to 784
instantiating 783
TDBComboBox 784 to 791
instantiating 791
TDBCtrlGrid 791 to 804
instantiating 802
TDBCtrlPanel 804 to 808
instantiating 808
TDBEdit 815 to 824
instantiating 822
TDBGrid 824 to 829
TDBImage 829 to 838
instantiating 837
TDBListBox 838 to 844
instantiating 844
TDBLookupComboBox 845 to 852
instantiating 851
TDBLookupControl 852 to 859
instantiating 858
TDBLookupListBox 859 to 865
instantiating 865
TDBMemo 867 to 875
instantiating 874
TDBNavigator 876 to 884
instantiating 883
TDBRadioGroup 884 to 891
instantiating 891
TDBRichEdit 891 to 901
instantiating 899
TDBText 901 to 906
instantiating 905
TDdeClientConv 906 to 916
instantiating 912
TDdeClientItem 917 to 921
instantiating 920
TDdeServerConv 921 to 925
TDdeServerItem 925 to 931
TDecisionCube 931 to 936
instantiating 934
TDecisionGraph 937 to 944
TDecisionGrid 944 to 949
TDecisionPivot 949 to 958
instantiating 957
TDecisionQuery 958 to 963
TDecisionSource 964 to 986
instantiating 974
TDirectoryListBox 986 to 996
instantiating 993
TDrawGrid 996 to 1006
TDriveComboBox 1007 to 1012
instantiating 1012
TDSTableProducer 1013 to 1027
instantiating 1022
technical support 2
TEdit 1027 to 1032
Template property
TCommonDialog 195
Temporary property
TDatabase 689
Terminate method
TApplication 45
Terminated property
TApplication 28
TerminateExtension property
TISAPIApplication 1163
text 1498
See also memos; text controls
copying 385
deleting 385
editing 376, 822, 867
editing *See* editing text
formatting 594
indenting 551
pasting 389
replacing 1393
selecting
edit controls 378, 379, 381, 385, 389, 390
masked edit controls 545, 547
rich text edit boxes 596, 598, 605, 606
wordwrapping 551, 557
text alignment 549, 556
captions 276, 1380
panels 564
labels 482
text attributes
controls 228, 229, 233
default 592
rich text edit controls 595
text boxes *See* edit controls
text buffers 253, 261
text controls 901
autosizing 902
creating/destroying 905, 906
getting values 903
text fields 1226, 1519
Text property
TControl 235
TCustomMaskEdit 539
TDBLookupComboBox 848
TDdeClientItem 919
TDdeServerItem 927
TField 1055
TQuery 1368
text strings *See* strings
TextCase property
TDriveComboBox 1009
tfBlob constant 690, 1461
tfBytes constant 1205
tfConnect constant 690, 1461
tfDataIn constant 1462
tfDataOut constant 1462
tfError constant 690, 1461
tfFrames constant 1205
tfHMS constant 1205
TField 1032 to 1065
instantiating 1060
TFileListBox 1065 to 1073
instantiating 1073
TFilterComboBox 1074 to 1080
instantiating 1080
TFindDialog 1080 to 1086
instantiating 1085
TFloatField 1086 to 1093

instantiating 1093
tfMilliseconds constant 1205
tfMisc constant 690, 1462
tfMSF constant 1205
TFontDialog 1093 to 1099
instantiating 1098
TForm 1099 to 1105
tfQExecute constant 690, 1461
tfQPrepare constant 690, 1461
tfSamples constant 1205
tfSMPTE24 constant 1205
tfSMPTE25 constant 1205
tfSMPTE30 constant 1205
tfSMPTE30Drop constant 1205
tfStmt constant 690, 1461
tfTMSF constant 1205
tfTransact constant 690, 1461
tfVendor constant 690, 1462
tgCustom constant 1311
tgEmbed constant 1311
tgImage constant 1310
tgImageMap constant 1310
tgLink constant 1310
tgObject constant 1311
TGraphicControl 1105 to 1109
instantiating 1109
TGraphicField 1110 to 1113
instantiating 1113
TGroupBox 1113 to 1117
tgTable constant 1310
THeaderControl 1118 to 1124
instantiating 1124
THintWindow 1125 to 1130
instantiating 1129
THotKey 1130 to 1134
Thousands property
TCustomUpDown 667
ThreadCacheSize property
TCustomServerSocket 607
three-dimensional
controls 1635, 1638
thumb tabs *See* scroll bars
TickMarks property
TTrackBar 1595
TickStyle property
TTrackBar 1596
tiDirtyRead constant 691
Tile method
TForm 1105
TileMode property
TCustomForm 401
TImage 1134 to 1140
instantiating 1140
TImageList 1141 to 1144
time 760, 768, 1036, 1563
formatting 764
Time property
TDateTimePicker 773
TimedScroll method
TCustomDBGrid 347
TCustomGrid 451
TimeFormat property
TMediaPlayer 1204
timer component 1567 to 1571
creating/destroying 1570, 1571
enabling/disabling 1568
timers (animation controls) 15
Timers property
TAnimate 15
TInplaceEdit 1144 to 1153
instantiating 1148
TIntegerField 1153 to 1160
instantiating 1160
tiReadCommitted constant 691
tiRepeatableRead constant 691
TISAPIApplication 1160 to 1163
title bars 394
common dialogs 1296
Title property
TApplication 28
TOpenDialog 1296
TWebApplication 1622
TitleClick method
TCustomDBGrid 348
TitleFont property
TCustomDBGrid 324
titles 28
See also captions; labels
TLabel 1164 to 1167
tlBottom constant 484
tlCenter constant 484
TListBox 1167 to 1172
TListView 1172 to 1177
TLoginDialog 1177 to 1182
tlTop constant 484
TMainMenu 1182 to 1187
TMaskEdit 1187 to 1191
tmBoth constant 1595
tmBottomRight constant 1595
TMediaPlayer 1191 to 1222
instantiating 1214
TMemo 1222 to 1226
TMemoField 1226 to 1230
instantiating 1230
TMenu 1230 to 1235
instantiating 1233
TMenuItem 1235 to 1247
instantiating 1245
tmTopLeft constant 1595
TNumericField 1247 to 1254
instantiating 1254
TObject 1255 to 1264
instantiating 1258, 1260, 1263
Toggle method
TCustomCheckBox 280
ToggleDimIndex method
TDecisionSource 985
toggles *See* check boxes
TOleContainer 1264 to 1284
instantiating 1278
TOleControl 1284 to 1291
instantiating 1288
tool windows 1586 to 1590
borders 1587, 1588
creating 1590
toolbars 567, 1571 to 1579
adding buttons 1574, 1580
captions 1576
context menus 1582
counting buttons 1573
creating/destroying 1579
disabling 1574
event handling 1577
images 1574, 1575, 1576, 1582
line wrap 1576
margins 1576
multiple rows 1576, 1583
removing buttons 1586
selecting buttons 1581, 1582, 1583
size 1573
styles 1583
transparent 1575
tooltips *See* Help Hints
Top property
TControl 235
TFindDialog 1083
ToPage property
TPrintDialog 1339
TOpenDialog 1291 to 1299
instantiating 1298
TOpenPictureDialog 1299 to 1302
instantiating 1302
TopIndex property
TCustomListBox 496
TopItem property
TCustomListView 516
TCustomTreeView 651
TopLeftChanged method
TCustomDecisionGrid 375
TCustomGrid 452
TDrawGrid 1006

topmost forms 43, 44
TopRow method
  TCustomGrid 434
Totals property
  TCustomDecisionGrid 365
TPageControl 1302 to 1309
  instantiating 1307
TPageProducer 1309 to 1312
TPaintBox 1312 to 1316
  instantiating 1316
TPaintControl 1316 to 1320
TPanel 1316 to 1320
TPasswordDialog 1321 to 1327
TPersistent 1327 to 1331
TPopupMenu 1331 to 1336
  instantiating 1335
TPrintDialog 1336 to 1341
TPrinterSetupDialog 1341 to 1343
TProgressBar 1343 to 1349
  instantiating 1348
TPropertyPage 1349 to 1355
TProvider 1356 to 1361
  instantiating 1360
TQuery 1361 to 1374
  instantiating 1372
TQueryTableProducer 1374 to 1378
TraceFlags property
  TDatabase 689
  TSession 1461
track bars 1591 to 1599
  adding tick marks 1595, 1596, 1599
  creating 1598
  event handling 1596
  incrementing/decrementing 1592
  moving through 1593, 1594
  orientation 1594
  position 1593, 1594
  selection ranges 1594, 1595
TrackLength property
  TMediaPlayer 1206
TrackPosition property
  TMediaPlayer 1206
Tracks property
  TMediaPlayer 1206
TRadioButton 1378 to 1383
  instantiating 1383
TRadioGroup 1383 to 1387
transactions 685
  batch processing 53
  committing 693, 695
  isolation levels 690
  rolling back 697
  starting 697
transfer appearance *See* drag image
TransIsolation property
  TDatabase 690
Translate method
  TDataSet class 743
Transliterate property
  TBatchMove 59
  TBlobField 114
  TStringField 1524
transparent images 465, 469, 473, 474
  colors 463
transparent labels 484
Transparent property
  TAnimate 15
  TCustomLabel 484
  TImage 1137
tree controls *See* tree views
tree nodes *See* tree views
tree views 646, 1599
  accessing nodes 649
  associating data with 661, 662
  border styles 648
  collapsing nodes 650, 653, 659
  creating/destroying 658, 659
  cursor position 660
  deleting nodes 654
  displaying nodes 650
  drop targets 648
  editing 654, 661
  event handling 652 to 655
  expanding nodes 649, 650, 654, 655, 659
  getting node position 651
  inserting bitmaps 648
    state images 651
  linking nodes 650, 651
  loading values 661, 662
  read-only 649
  saving 662
  selecting nodes 648, 652, 661
  sorting nodes 651, 653, 658
trees *See* tree views
TRemoteServer 1388 to 1393
TReplaceDialog 1393 to 1396
  instantiating 1396
TReport 1396 to 1408
  instantiating 1404
TRichEdit 1408 to 1413
trimming *See* truncating
TrueType fonts 1096
truncating data 52, 57
tsAuto constant 1596
TSaveDialog 1413 to 1415
TSavePictureDialog 1415 to 1417
TScreen 1418 to 1430
  instantiating 1429
TScrollBar 1430 to 1438
  instantiating 1437
TScrollBox 1438 to 1443
  instantiating 1443
TScrollingWinControl 1443 to 1451
  instantiating 1449
TServerSocket 1451 to 1454
  instantiating 1454
TSession 1454 to 1476
  instantiating 1467
TShape 1476 to 1480
  instantiating 1480
TSmallIntField 1481 to 1485
  instantiating 1484
tsManual constant 1596
tsNone constant 1596
TSpeedButton 1485 to 1492
  instantiating 1492
TSplitter 1493 to 1498
  instantiating 1497
TStaticText 1498 to 1502
TStatusBar 1502 to 1508
  instantiating 1508
TStoredProc 1508 to 1519
  instantiating 1517
tsTrackBegin constant 1122
tsTrackEnd constant 1122
tsTrackMove constant 1122
TStringField 1519 to 1526
  instantiating 1526
TStringGrid 1526 to 1533
  instantiating 1533
TTabControl 1533 to 1537
TTable 1538 to 1558
  instantiating 1550
TTabSheet 1558 to 1563
  instantiating 1563
ttASCII constant 1546
ttDBase constant 1546
ttDefault constant 1546
TTimeField 1563 to 1567
  instantiating 1567
TTimer 1567 to 1571
  instantiating 1570
TToolBar 1571 to 1579
  instantiating 1579

TToolButton 1580 to 1586
instantiating 1585
TToolWindow 1586 to 1590
instantiating 1590
ttParadox constant 1546
TTrackBar 1591 to 1599
instantiating 1598
TTreeView 1599 to 1604
TUpdateSQL 1604 to 1611
TUpDown 1611 to 1615
TVarBytesField 1615 to 1619
instantiating 1618
TWebApplication 1619 to 1626
instantiating 1624
TWebDispatcher 1626 to 1628
TWebModuler 1629 to 1631
TWinControl 1631 to 1675
instantiating 1649
two-dimensional controls 1635, 1638
TWordField 1675 to 1679
instantiating 1679
types *See* data types; field types
typography 2

U

uaAbort constant 80, 81
uaApplied constant 80, 81
uaFail constant 80, 81
uaRetry constant 80, 81
uaSkip constant 80, 81
udHorizontal constant 666
udLeft constant 665
udRight constant 665
udVertical constant 666
ukDelete constant 79, 81, 163, 1358, 1607, 1608, 1609, 1610
ukInsert constant 79, 81, 163, 1358, 1607, 1608, 1609, 1610
ukModify constant 79, 81, 163, 1358, 1607, 1608, 1609, 1610
undoing data changes 90, 178, 181
UndoLastChange method
TClientDataSet 181
unhandled exceptions 30
UnhookMainWindow method
TApplication 46
UniDirectional property
TQuery 1369
unique indexes 1548
unlocking tables 1557
UnlockTable method
TTable 1557
Unmerge method
TMainMenu 1186
UnPrepare method
TQuery 1374
TStoredProc 1519
unpublished properties 248, 1330
UnRegisterChanges method
TCustomImageList 481
update controls
specifying 76
Update method
TControl 262
TDirectoryListBox 995
TFileListBox 1073
TWinControl 1674
update objects 747 to 749
UpdateBoundsRect method
TControl 262
UpdateColumn method
TCustomListView class 533
UpdateColumns method
TCustomListView 533
UpdateContents method
TInplaceEdit 1152
UpdateControlState method
TWinControl 1674
UpdateCursorPos method
TDataSet 743
Updated method
TComponent 215
TMediaPlayer 1221
UpdateDesigner method
TCustomGrid 452
UpdateFormatSettings property
TApplication 28
UpdateItems method
TCustomListView 533
UpdateLoc method
TInplaceEdit 1152
UpdateLock property
TCustomDBGrid 324
UpdateMetricSettings property
TApplication 29
UpdateObject method
TOleContainer 1283
TPropertyPage 1355
UpdateObject property
TBDEDataSet 76
UpdatePropertyPage method
TPropertyPage 1355
UpdateRecord method
TDataSet 744
UpdateRecordTypes property
TBDEDataSet 77
UpdateRegistry method
TComponent 215
UpdatesPending property
TBDEDataSet 77
UpdateStatus method
TBDEDataSet 91
TClientDataSet 181
UpdateVerbs method
TOleContainer 1284
Updating method
TComponent 215
up-down controls 663, 1611
companions 665
creating 671, 672
event handling 668
incrementing/
decrementing 666, 667
keyboard processing 665
message processing 672
orientation 666
position 664, 666, 667, 671
uppercase characters 379
user options 141, 275, 777
mutually exclusive 1378

V

Validate method
TField 1064
ValidateContainer method
TComponent 216
ValidateEdit method
TCustomMaskEdit 547
ValidateError method
TCustomMaskEdit 548
ValidateInsert method
TComponent 216
ValidateName method
TDatabase 697
ValidateRename method
TComponent 216
TCustomForm 422
ValidChars property
TField 1055
ValidFieldIndex method
TCustomDBGrid 348
validity checking 534
error handling 548
Value property
TBCDField 67
TBinaryField 98
TBlobField 114
TBooleanField 123
TDateTimeField 766
TDBRadioGroup 888
TField 1055
TFloatField 1091

TIntegerField 1158
TStringField 1524
ValueChecked property
TDBCheckBox 780
Values property
TDBRadioGroup 888
ValueUnchecked property
TDBCheckBox 780
variable-length bytes fields 1615
creating 1618
variants 1037
binary data as 97, 112
Boolean values as 122
dates as 764
numbers as 66, 1089, 1156
OLE objects 1271
reassigning 1257
strings as 1523
summary values as 314, 316, 975, 976, 980
VCLComObject property
TComponent 202
VCRs *See* media players
VersionMajor property
TReport 1401
VersionMinor property
TReport 1402
Vertical property
TCoolBar 268
vertical scroll bars 1432, 1445
grids 433
vertical track bars 1594
VertScrollBar property
TScrollingWinControl 1445
video clips 8
videos *See* media players
ViewOrigin property
TCustomListView 516
ViewStyle property
TCustomListView 517
Visible event
TField 1056
TMenuItem 1243
Visible property
TControl 236
TCustomForm 401
VisibleButtons property
TDBNavigator 879
TMediaPlayer 1207
VisibleChanging method
TControl 262
TCustomForm 422
VisibleColCount method
TCustomGrid 434
VisibleRowCount event
TCustomListView 517
VisibleRowCount method
TCustomGrid 434
visual components 217
vsIcon constant 517
vsList constant 517
vsReport constant 517
vsSmallIcon constant 517
vtAnsiString constant 1060
vtBoolean constant 1060
vtChar constant 1060
vtCurrency constant 1060
vtExtended constant 1060
vtInteger constant 1060
vtPChar constant 1060
vtString constant 1060
vtVariant constant 1060

W

Wait property
TMediaPlayer 1207
WaitStat property
TDdeClientConv 910
WantReturns property
TCustomMemo 551
WantTabs property
TCustomMemo 551
Web dispatcher 295, 673, 1626
accessing items 674, 677, 678
creating/destroying 677, 678
event handling 675, 676
handling requests 674, 678
handling responses 675, 679
message processing 680
Web module 1629 to 1631
Web pages 1309
content producers 295
HTML templates 557
Web server applications 138, 1160, 1619 to 1626
action items 674, 677, 678, 1629
active connections 1620
cached connections 1620, 1621
creating 1624, 1626
destroying 1625
exception handling 1625
initializing 1626
minimizing 1622
request transactions 1621, 1623, 1624, 1625
running 140, 1626
Web site 2
Width property
TControl 236
TCustomImageList 466
TScreen 1427
Win-CGI programs *See* CGI applications
window handles *See* handles
windowed controls 263, 298, 1631 to 1675
accessing 1640, 1647, 1658, 1660, 1671
adding children 1661
aligning 1647, 1654, 1657
boundaries 1672
creating 1649, 1650, 1651, 1652
displaying 1639, 1673, 1674
event handling 1641 to 1643, 1654, 1655, 1656, 1662, 1663
getting children 1634, 1648, 1649, 1657, 1658, 1671, 1672
location 1658
origins 1633
message processing 1635, 1647, 1652, 1662, 1664, 1665, 1675
painting 1633, 1661, 1665, 1666, 1668, 1674
removing children 1668
scaling 1648, 1670
scrolling 1671
size 1634, 1659
style 1635, 1638
tab order 1639
WindowHandle property
TMenu 1232
TWinControl 1640
WindowMenu property
TCustomForm 401
WindowProc property
TControl 236
Windows common dialogs *See* common dialog boxes
WindowState event
TCustomForm 402
WindowText property
TControl 237
.WMF files *See* metafiles
WndProc method
TButtonControl 134
TControl 263
TCustomComboBox 294
TCustomForm 423
TCustomListBox 506
TCustomListView 534

TDBMemo 875
TInplaceEdit 1153
TWinControl 1675
word field types 1675
creating 1679
WordWrap method
TCustomLabel 485
WordWrap property
TCustomMemo 551
wordwrapping 485, 551, 557
working directories 684
Wrap property
TCustomUpDown 667
TToolButton 1583
Wrapable property
TToolBar 1576
WriteState method
TComponent 216
wsMaximized constant 402
wsMinimized constant 402
wsNormal constant 402

X

XNS connections 619
xtCheck constant 965
xtColumns constant 955
xtHorizontal constant 954
xtLeftTop constant 954
xtRadio constant 965
xtRadionEx constant 965
xtRows constant 954
xtSummaries constant 955
xtVertical constant 954

Z

z-order 261, 1673

Borland
www.borland.com

Copyright © 1997 Borland International, Inc. All rights reserved. All Borland product names are trademarks of Borland International, Inc. Corporate Headquarters: 100 Borland Way, Scotts Valley, CA 95066-3249, (408) 431-1000. Internet: http://www.borland.com CompuServe: GO BORLAND. Offices in: Australia, Canada, France, Germany, Hong Kong, Japan, Latin America, Mexico, The Netherlands, Taiwan, and United Kingdom • Part # HDA1330WW21774 • BOR 9989